COMMODORE 64

file–kezelés és input–output

LSI Alkalmazástechnikai Tanácsadó Szolgálat

intelorg

DBASIC V1.2

A DBASIC az egyetlen adatfeldolgozási célra kifejlesztett interpreter. Jelentősen megnöveli a gép hatékonyságát:

- a használható RAM terület kibővül a ROM-ok alatti RAM-ok használatával (16 Kbyte)
- numerikus mezők használata esetén a diszkette tárolási kapacitása kétszeres
- a programok futási ideje jelentősen rövidül
- a string-kezelés átírásával megszünik a gyakran több perces várakozás, elmarad a "szemétgyűjtő" eljárás

Jelentősen megnő a programozás hatékonysága:

- azonos feladatot ellátó program elkészítési ideje az interpreterrel töredéke a BASIC programozás idejének
- szükségtelenné válik assembler rutinokkal a programozást támogatni, az adatfeldolgozás során felmerülő rutinokat a DBASIC utasításként tartalmazza.

Jelentősen csökken a feladatok megoldásának költsége. A forgalmazott - és csak egy-egy részfeladatot megoldó - alap software-k megvásárlása feleslegessé válik. Az ezek által elvégzendők a DBASIC-ben néhány utasítással megírhatók. (SORT, indexelt szekvenciális file-kezelés, közvetlen diskette I/O, stb.)
A már BASIC-ben megírt programok az interpreterrel változtatás nélkül futtathatók, a hatékony string-kezelés miatt a feldolgozás így is felgyorsul.

GYORSSZOLGÁLAT!

Felhívjuk a figyelmet új szolgáltatásunkra!
Ha már tudják, MIT kívánnak C 64-en megvalósítani, hívjanak bennünket!
Munkatársaink kiszállnak telephelyükre és a számítógépes rendszer első működő változatát (a "deszkamodellt") NÉHÁNY NAP alatt elkészítik.
Természetesen az ÖNÖK KÖZREMUKÖDESÉVEL.
A feladat megfogalmazásától a kész rendszer bevezetéséig csak néhány hét telik el!
A rendszer az Önök szeme előtt készül, ha igényeik menet közben változnak, azt AZONNAL beépítjük.

INTELORG GMK
Budapest, Dohány u. 58-62.
1077 Tel: 343-999

COMMODORE 64

FILE-KEZELÉS és INPUT-OUTPUT

LSI Alkalmazástechnikai Tanácsadó Szolgálat, Budapest, 1986

Írták: Farkas Zoltán

Bálint Balázs

Lektorálta: Dr. Úry László

Szerkesztette: Székely László

TARTALOMJEGYZÉK

BEVEZETÉS

Az emberi kapcsolatteremtés fajtái bizonyára összeszámlálhatatlanok: kezdve a szem ráncolásától az ököl rázásán és az írásbeli kommunikáción át végezve a digitalizált telefonüzenetig. Így van ez a számítógépekkel is: melyik, hogyan, milyen széles skálán képes környezetével és más számítógépekkel kommunikálni: ez a döntő kérdés. Hasonlítsuk csak össze az Apple II és Commodore 64 számítógépeket! Vagy vegyük a Sinclair Spectrumot! Ennek processzora a Z 80 sokkal fejlettebb és gyorsabb is mint a Commodore 64 látszólag teljesen szerény M 6510-ese, de a billentyűzet, a perifériaválaszték és nem utolsó sorban a video chip végülis az utóbbi javára dönti el a játszmát. (A félprofesszionális kategóriában legalábbis.)

A Commodore 64 a legelterjedtebb olyan mikrogép hazánkban, amit - a home computer funkción kívül - adatfeldolgozási, ügyvitelgépesítési feladatokra használnak. Ennek oka, hogy - árához képest - a beépített hardware igen jó, és a géphez a többi hasonló kategóriájú személyi számítógéphez képest jobb és megbízhatóbb lemezegységek és nyomtatók csatlakoztathatók. Elterjedésével kapcsolatos, hogy sok alap- és felhasználói software "könnyen" beszerezhető hozzá.

A Commodore 64-gyel kapcsolatban gyakran említik, hogy adatfeldolgozási feladatokra nem igazán alkalmas. Könyvünk azoknak próbál segítséget nyújtani, akik a gépet mégis adatfeldolgozási célokra kívánják használni.

Ez a könyv egyrészt a Commodore 64 I/O rendszerével, másrészt a Commodore file-szervezésével foglalkozik, olyan mélységben, ami az általánosságban teljességre törekvő kézikönyvekben nem található meg, tehát nem kezdőknek íródott (leszámítva az üzembe állításról szóló utolsó fejezetet). Feltételezzük legalább a BASIC (nem feltétlenül Commodore) alapos ismeretét. Persze akik nem "félnek" az assembler programozástól, munkájukhoz a könyvből sok segítséget kapnak.

A BASIC utasítások közül csak az I/O utasításokra térünk ki. Ezeket részletesen (referencia szinten) tárgyaljuk.

A Commodore gépcsalád a BASIC-ben és annak megvalósításában is közel homogén, de a Spectrum, vagy a HT 1080 Z számítógépekhez képest egyedi eljárást használ.

Az egyes perifériák kezelésére egyaránt az

```
OPEN
PRINT#
CMD
INPUT#
CLOSE
```

utasításokat használja.

A perifériák megkülönböztetésére az utasítás paramétereiben megadandó **egységszámot** (device number) és **csatornaszámot** (mádoslagos cím, secondary address) használja. A legfontosabb ROM-ban tárolt rutinok egy ugrótábla segítségével érhetők el, amelyek közül egyesek a RAM-ban lévő belépési ponton át indulnak. Ez nagyon megkönnyíti az I/O rendszer módosítását, illetve gépi kódból való felhasználását. Ez a szervezés a 8000-es sorozat gépeire éppúgy jellemző, mint a VC 20-ra!

Azt a területet, amire a gépet eredetileg tervezték (játékok, színes grafika, hangeffektusok) nem is érintjük.

A könyv egyes fejezeteinek mélysége eltérő. Néhány olyan témakört, mely ugyan szorosan kapcsolódik az itt tárgyalthoz, de más itthon is megjelent irodalom részletez, csak röviden, vagy egyáltalán nem ismertetünk (pl. hardware felépítés, adatbáziskezelő software-ek, stb.). Ebben a tárgykörben bőséges irodalom áll rendelkezésre:

Dr. Úry László : Commodore 64 Basic és felhasználói kézikönyv I-II.
(LSI ATSz 1984)

Erdős Iván : Commodore 64 Assembly
(LSI ATSz 1985)

Dr. Ferenczy A.: C-64 start! Alapfokon a Commodore 64-ről.
(LSI ATSz 1986)
Az Easy-file-tól a Master 64-ig
adatfeldolgozó programcsomagok C 64-en (LSI ATSz 1986)

1001 Játék és Graphics Basic Commodore 64-en
(LSI ATSz 1985)
CP/M (LSI ATSz 1985)
FORTH (LSI ATSz 1985)

Kimaradt a könyvből a kazettás egység (Datasette) tárgyalása, mert igényesebb munkánál lassúsága és megbízhatatlansága miatt - nem használható. A beépített RS 232 vonali csatlakozó használatát sem tárgyaljuk részletesen,mert ez elég speciális terület és részletes tárgyalása kimerítene egy egész könyvet.

Az első rész a BASIC és a KERNAL I/O utasitások, ill. gépi kódú rutinok leirását tartalmazza, egészen az IEC busz használatáig.

A második rész a file-szervezés kérdéseivel foglalkozik. Nem csak a VC 1541-es lemezegység beépitett lehetőségeit találhatjuk meg, hanem azokat a file-szervezési eljárásokat is, amelyek a gép kapacitását még nem haladják meg. Részletesen tárgyaljuk az indextáblák elhelyezésének lehetőségeit.S találunk - igaz csak egy példaprogram részletezettségével - egy személyzeti nyilvántartó programot is. Könyvünk mindazok számára tartogat új ismereteket, programozói fogásokat, akik a Commodore 64-gyel foglalkoznak. Az " I "-vel keretezett bekezdések a témával most ismerkedők számára iródtak.

A könyvben szereplő példákat ellenőriztük. Ha ennek ellenére maradt valahol hiba, ezért Olvasóink szíves elnézését kérjük!

Olvasóink minden, a könyvvel kapcsolatos megjegyzését előre is köszönjük.

A KIADÓ

BASIC I/O UTASÍTÁSAI

BEVEZETÉS

A COMMODORE 64-et tervezői olcsó otthoni számítógépnek szánták. Az olyan perifériákat, melyek nem tartoznak szorosan az alap konfigurációhoz (floppy) külön, a periféria ROM-ban lévő software működteti (intelligens periféria). Ezek a perifériák (leegyszerűsítve a folyamatot) a következő elven működnek:

- A C-64 megcímzi a perifériát, majd egy BASIC I/O utasítás egy parancsstringet küld ki a buszon. A parancs szintaxisát az interpreter ellenőrzi.
- A perifériákat működtető software ellenőrzi a parancs szintaxisát. Ha az hibátlan, végrehajtja és jelzi a C-64-nek, a végrehajtást, illetve a hibát, ha nem végrehajtható.
- A C-64 fogadja a végrehajtás vagy a hiba jelzését.

Az utasítások tárgyalásánál csak a C-64 interpreter és a KERNAL működésére térünk ki. A C-64 által kezelt "intelligens" perifériák tevékenységét (a C-64 által küldött parancsstring hatására) az egyes készülékeknél tárgyaljuk.

OPEN

Feladata: file megnyitása

Szintaxis: OPEN lf [,ks][,cs][,p]

lf: logikai fileszám
értékének az 1-255 intervallumba kell esnie

ks: készülékszám
értékének a 0-255 intervallumba kell esnie

cs: csatornaszám (használatos elnevezések még: másodlagos cím, megnyitási mód)
értékének a 0-255 intervallumba kell esnie

p: parancs vagy filenév
a parancs "hatását" és szintaxisát a "PERIFÉRIÁK" fejezetben készülékenként tárgyaljuk

Az utasítás végrehajtása

a/ Paraméterek beolvasása és kiértékelése

A BASIC interpreter a hiányzó paramétereket alapértelmezés szerinti értékkel helyettesíti:
- készülékszám alapértelmezése: 1
- csatornaszám alapértelmezése: 0, ha a készülékszám kisebb, mint 4
  255, ha a készülékszám nagyobb,mint 3
- parancs alapértelmezése : üres string

Kiértékelés közben a paraméter értékek a megfelelő rendszerváltozóba töltődnek:

$b8	:	logikai fileszám
$b9	:	csatornaszám
$ba	:	készülékszám
$bb/bc	:	mutató: parancs string címe
$b7	:	parancs string hossza

b/ File elhelyezése a nyitott file-ok táblázatában

Egy nyitott file-t nem lehet mégegyszer megnyitni. A nyitott file-ok számát a $98 rendszerváltozó tartalmazza. A KERNAL a nyitott file-ok paramétereit három táblázatban tartja nyilván:
- logikai file-számok: $0259 - $0262
- készülékszámok : $0263 - $026c
- csatornaszámok : $026d - $0276

(Az éppen megnyitás alatt álló file száma nem szerepelhet a logikai fileszámok táblázatában.)

c/ A készüléktől, csatornaszámtól és a parancstól (file-névtől) függő egyéb tevékenység:

- Ha a kijelölt készülék a billentyűzet vagy a képernyő, nincs további tevékenység.
- Ha a készülékszám nagyobb, mint 3, a KERNAL ezt a soros buszként értelmezi, a buszra kapcsolt perifériákat egységesen kezeli.
  Ha a parancs üres string, nincs további tevékenység. (A buszra fel sem fűzött, ill. nem bekapcsolt készülék kijelölése esetén hibaüzenetet nem kapunk.)
  Ha a parancs nem üres string, a KERNAL megszólítja a kijelölt készüléket (LISTEN-lásd a KERNAL c. fejezetet).
  Ezt követi az OPEN kód (240 + csatornaszám alacsony félbyte), majd a parancs (file-név) átvitele (kiadása a soros buszra).
  Az átvitelt UNLISTEN (lásd a KERNAL c. fejezetet) zárja le.
- Ha a kijelölt készülék a kazettás egység, a tevékenység a csatornaszámtól függ, ami az input vagy output file-t jelöl ki (lásd táblázat).
  INPUT: - várakozás a PLAY gomb megnyomására
  - "searching" ("for name") üzenet kiírása
  - ha a parancsstring nem üres (definiált file-név) a keresett file-név, egyébként a következő blokk header keresése
  - ha a szalagon az első file megvan (definiált file-név esetén a keresett file), az utasítás befejeződik.
  OUTPUT:- várakozás a RECORD és a PLAY gomb
  - megnyomására
  - header felírása a szalagra
  - megjegyzi, kell-e EOT blokkot írni a file lezárása után

- Ha a kijelölt készülék az RS232 vonal. A tevékenységet csak vázlatosan adjuk meg (könyvünkben az RS232 vonallal nem foglalkozunk részletesen), kiemelve a megnyitás "hatását" a BASIC területre
  - parancsstring tárolása
  - parancsstring kiértékelése: átviteli sebesség, status beállítása, I/O pufferek kijelölése
  - BASIC terület végének beállítása
  A kijelölt két puffer 256 byte-os, a BASIC terület korábbi végénél helyezkedik el. Az OPEN utasítás a BASIC RAM végét két blokkal (512 byte) csökkenti, ezt egy CLR végrehajtásą követi. (A programban eddig használt változók értéke "elveszik". A megnyitott file-ok, az RS232-t kivéve törlődnek.)

A készülékszám és a csatornaszám értelmezése

A készülékszám és a csatornaszám definiciójánál láttuk, hogy értéküknek a 0-255 intervallumba kell esnie.
A file megnyitását végző KERNAL rutinok a csatornaszám értékéből csak az alsó félbyte-ot használják. Ez alól egy kivétel van: ha a kijelölt készülék a soros buszra felfűzött (a készülékszám nagyobb mint 3) és a csatornaszám nagyobb, mint 127, a parancsstring átvitele (kiadása a soros buszra) nem történik meg.
Az utasítás végrehajtásakor a szintaxist kiértékelő BASIC rutin és a készüléket kezelő KERNAL rutinok a megadott készülékszám értelmezhetőségét nem vizsgálják (pl. nagyobb, mint 11). Ha egy nem használatos készülékszámra hivatkozunk (pl. OPEN1,142,15,"I") a hibaüzenet megegyezik az egyébként használatos, de nem bekapcsolt készülékre hivatkozással.

A használatos készülékszámok és csatornaszámok:

Készülékszám	Megcímzett készülék	Csatornaszám
0	billentyűzet	nincs értelmezve
1	kazettás egység	0: input 1: output 2: output EOT blokk írással
2	RS232	nincs értelmezve
3	képernyő	nincs értelmezve
soros busz		
4,5	nyomtató(soros buszon)	a használható értékek típustól függenek
8-11	floppy egység	0: load 1: save 2-14: szabadon használt adat csatorna 15: parancs csatorna

Az utasítás végrehajtásakor lehetséges hibaüzenetek

- "Syntax" (BASIC)
A szintaxis hibás
Pl. OPEN3, (logikai file-szám után írt "," további paramétert jelez, de nincs megadva több paraméter).

- "File open" (KERNAL)
A hivatkozott logikai file-szám már szerepel a logikai file-számok táblázatában.

- "Too many files" (KERNAL)
Már 10 file-t definiáltunk korábban.

- "Illegal device number" (KERNAL)
Csak egy esetben léphet fel: a kijelölt készülék a kazettás egység és a puffer címe kisebb, mint $200

- "File not fund" (KERNAL)
A kijelölt készülék a kazetta, input-ként megnyitva és a parancsstringben definiált nevű szalagheader a megnyitott szalagon nincs.

- "Device not present" (KERNAL)
Az OPEN utasítás a soros buszra hivatkozik, a megnyitáskor parancsstringet is megadtunk és a megcímzett készülék nincs a buszon (nincs bekapcsolva), ill. a kijelölt csatornán nem lehetséges az I/O.

- "Not input file" (KERNAL)
A megadott logikai file-szám 0. (Miért pont ez az üzenet?)

- "Illegal quantity" (BASIC)
A megadott logikai file-szám, készülékszám, vagy csatornaszám nem esik a 0-255 intervallumba.

Példák:

OPEN 1
Az 1-es logikai file-számmal megnyitottuk a kazettás egységen inputként az éppen következő file-t.

OPEN2,3,8,"VALAMI"
A 2-es logikai file-számmal megnyitottuk a képernyőt. (A csatornaszámot és a parancsstringet az interpreter figyelmen kívül hagyja.)

OPEN3,8,15
A 3-as logikai file-számmal megnyitottuk a 8-as floppy parancs csatornáját.

OPEN3,8,143
Hatása ugyanaz, mint az előző példában.

OPEN4,2,139
A 4-es logikai file-számmal megnyitottuk az RS232 vonalat, a kontrolszó az előbbiekben beállított (most parancsstringként nem adtuk meg).A csatornaszámot az interpreter figyelmen kívül hagyja.

OPEN5,8,3,"FILE"
Megnyitottunk a 8-as floppy egységen egy "FILE" nevű file-t, inputként.

OPEN5,8,3,"FILE,W"
Megnyitottunk a 8-as floppy egységen egy "FILE" nevű file-t, outputként.

OPEN5,48
Megnyitottuk a 48-as számmal azonosított készüléket. A 48 nem használt készülékszám. A szintaxis kiértékelése után a megnyitás végrehajtásáról előtét assembler rutinnal kell gondoskodnunk, különben a megnyitás hatástalan. A KERNAL OPEN rutinja ($f34a) a 3. lapon lévő $031a címen keresztül átírható.

CLOSE

Feladata: file lezárása

Szintaxis: CLOSE lf [,ks] [,cs] [,p]

A szintaxis az OPEN utasítással megegyező, mert az interpreter a kiértékeléshez ugyanazt a rutint használja. A logikai file-számon kívül a többi paraméter felesleges és "hatástalan", a végrehajtó KERNAL rutin semmire sem használja őket.

Az utasítás végrehajtása

a/ Logikai file-szám keresése
A logikai file-számot egy KERNAL rutin megkeresi a nyitott file-ok táblázatában (lásd OPEN utasítás). Ha a file-szám nem szerepel a táblázatban, további tevékenység nincs.

b/ File paraméterek beolvasása
A file-számhoz rendelt készülékszám és csatornaszám a táblázatból a megfelelő rendszerváltozókba töltődik:
$b8: logikai file-szám
$b9: csatornaszám
$ba: készülékszám

c/ A készüléktől függő egyéb tevékenység
- Ha a készülékszám nagyobb, mint 3 (soros busz):
  - készülékszám kiadása a buszra
  - LISTEN (lásd KERNAL c. fejezet)
  - CLOSE kód (224 + csatornaszám alsó félbyte) kiadása a buszra
  - UNLISTEN (lásd KERNAL c. fejezet)

Ha a lezárt file output file volt, a kijelolt készülék további tevékenységet végez.

- Ha a kijelölt készülék a kazettás egység és a file outputként lett megnyitva
  - puffer tartalom kiírása (ha a puffer nem üres)
  - ha a csatornaszám = 2, EOT blokk felírása
- Ha a kijelölt készülék az RS232 vonal
  - I/O pufferek törlése
  - BASIC terület végének újra beállítása, CLR (lásd OPEN utasítás)
- Ha a kijelölt készülék a billentyűzet, a képernyő vagy a kazettás egység (inputként megnyitva), a KERNAL-nak nincs tevékenysége.

d/ File leíró táblázatok újra írása

- A nyitott file-ok számát mutató rendszerváltozó ($98) értéke eggyel csökken.
- A file leíró táblázatokból a lezárt file paraméterei törlődnek. A táblázatban az aktív file-ok paraméterei egy hellyel előbbre kerülnek.

Megjegyzés:
A CLR utasítás töröl minden megnyitott file-t (törli a file leíró táblázatokat, de a floppyn nem zárja le a file-t)

Példák:

CLOSE1
Lezárja az 1-es logikai file-számmal megnyitott file-t.

CLOSE1,138,47,"MINDEGY"
Lezárja az 1-es logikai file-számmal megnyitott file-t.

CLOSE1,2,3
Lezárja az 1-es logikai file-számmal megnyitott file-t. (Csak az 1-es file-t zárja le, a 2 és 3 a készülékszám és csatornaszáma, ezt a KERNAL nem használja!)

PRINT CMD PRINT#

Feladatuk: nyomtatás a megadott készülékre

A három utasítást együtt tárgyaljuk, mert az output készülék kijelölésétől eltekintve működésük azonos.

Szintaxis:
```
PRINT [ listaelem ] [ listaelem ] ....
PRINT#lf [ ,listaelem ] [ listaelem] ....
CMDlf [ ,listaelem] [ listaelem] ....
```

lf: logikai file-szám
Egy előzőleg sikeresen megnyitott output file.
A file-számnak az 1-255 intervallumba kell esnie.

Listaelem: string kifejezés
aritmetikai kifejezés
tabulálás (TAB)
helyköz funkció (SPC)
kifejezés terminátor

Listakép: listaelemek sorozata

String kifejezés

String változókat, string konstansokat, string operátorokat és stringkezelő BASIC függvényeket tartalmazó kifejezés. (Értéke tetszőleges ASCII értéksorozat lehet.)

String konstans: két " jel közötti tetszőleges hosszúságú ASCII értéksorozat

pl. "aaaa"
pl. " Lc qQE "
pl. "   KONSTANS "

Stringkezelő BASIC függvény: STR$
CHR$
LEFT$
RIGHT$
MID$

A funkció által kijelölt művelet végrehajtása után a művelet eredménye mint string konstans viselkedik.

pl. LEFT$(C$,3)
pl. "cdaac#"+MID$(Q$,4,5,)
pl. "a"+CHR$(129)+CHR$(X)+B$

Aritmetikai kifejezés

Numerikus változókat, numerikus konstansokat, numerikus operátorokat és numerikus BASIC függvényeket tartalmazó kifejezés.

Numerikus konstans: valós vagy egész konstans (megadható lebegőpontos formában is).

pl. 124
pl. 12.3 E+1
pl. 1234567.234
pl. .   (értéke: 0)

Numerikus BASIC függvény: PEEK, LEN VAL, ASC
COS, SIN, TAN, ATN
SGN, INT, ABS, SQR
RND, LOG, EXP
USR, POS

A funkció által kijelölt művelet végrehajtása után a művelet eredménye mint numerikus konstans lista elem viselkedik.

pl. ABS(A)
pl. LEN(M$)

Példák aritmetikai kifejezésre:

pl. 128 + PEEK(2) +0.4 -INT(A/10)+4
pl. A+B-C+C+(Q*3000)/28
pl. SIN(SIN(LOG(X)))

Tabulálás

a/ TAB (aritmetikai kifejezés)
Tabulálás az aritmetikai kifejezésben megadott oszlop pozícióra.
Az aritmetikai kifejezés értékének a 0-255 intervallumba kell esnie.
Ha a kurzor aktuális oszlop pozíciója nagyobb vagy egyenlő, mint a TAB funkcióban megadott érték, a tabulálás nem hajtódik végre. Egyébként tabulálás történik a megadott értékig.
Ha az otuput készülék a képernyő, a tabulálás kurzor jobbra vezérlő karakterek kiadását jelenti a megadott oszlop pozícióig, egyébként az aritmetikai kifejezésben megadott számú szóköz átvitele történik. (Ha az output készülék képernyő vagy nyomtató, a pozícionálás esetleg több sorra előre is történhet.)

pl. TAB(128-A+PEEK(438))
pl. TAB(0) (hatástalan)

b/ /,/
10 pozíciós tabulátor
Tabulációs pontok 0, 10, 20, 30.
Ha az otuput készülék a képernyő, a tabulálás kurzor jobbra vezérlő karakterek kiadását jelenti a következő tabulációs pontig.
Ha az output készülék nem a képernyő, a tabuláció hatása: 10 helyköz karakter átvitele az output készülékre (megegyezik az SPC(10) helyköz funkcióval).

pl. A$,549,VAL(X$)
pl. A+B,B$,COS(28),EQ$

Helyköz funkció

SPC (aritmetikai kifejezés)
Az aritmetikai kifejezésben megadott számú kurzor jobbra vezérlő karakter (ha az output készülék a képernyő), ill. szóköz (ha az output készülék nem a képernyő) átvitele az output készülékre.
Az aritmetikai kifejezés értékének a 0-255 intervallumba kell esnie.

pl. SPC(ASC(G$))
pl. SPC(45)
pl. SPC(129-A/12)

Kifejezés terminátor: /;/

Lista elemek elválasztására használjuk.
A kifejezés terminátor csak a BASIC interpreter kifejezés kiértékelő rutinja miatt szükséges. Ha egy lista elemet a rutin számára terminátorként értelmezhető karakter zár le, a kifejezés terminátorra nincs szükség, ill. használata a lista képre nincs hatással.
A kifejezés terminátornak (;) a listaképre csak utolsó listaelemként van hatása. Ha az utolsó elem nem /;/, az interpreter egy CHR$(13) karakterrel "megtoldja" a listaképet.

```
pl. PRINT A$PEEK(500)(639)ACOS(28)
          ↑         ↑    ↑
  string változó  BASIC  konstans
   nevének vége  funkció  vége
                  vége
pl. PRINT A+28-D;C;129
                ↑ ↑
```

itt szükséges a terminátor, mert terminátorok nélkül a DC változó értéke tartozna hozzá az aritmetikai kifejezéshez.

Az utasítások végrehajtása

A PRINT# utasítás

- A megadott logikai file megnyitásakor a file-hoz rendelt készüléket jelöli ki output készüléknek.
- A listakép kiértékelése.

A végrehajtás listaelemenként történik. Ha a listaelem tabulációs funkció, helyköz funkció, vagy kifejezés terminátor, közvetlenül megtörténik a megfelelő funkció végrehajtása. Más listaelem esetében a BASIC interpreter kifejezés kiértékelő rutinja működik. Ha a listaelem numerikus típusú, a kiértékelést stringgé alakítás követi. Ezután következik a string átvitele az output készülékre, karakterenként.

Ha a listaelem numerikus típusú volt, ezután egy kurzor jobbra (ha az output készülék nem a képernyő) átvitele következik.

Ha az utolsó elem nem /;/, a végrehajtást egy "carriage return" átvitele zárja le.

- Aktív output csatorna törlése. (Visszaállítja az alapértelmezést, output készülék ismét a képernyő.)

A PRINT utasítás

- A listakép kiértékelése. Lásd PRINT# utasításnál.

A CMD utasítás

- A megadott logikai file megnyitásakor a file-hoz rendelt készüléket jelöli ki output készüléknek. (Ezután a rendszer üzenetei ide kerülnek, ill. az ezt követő PRINT vagy LIST utasítás esetén is ide kerül az output.)
- A listakép kiértékelése. Lásd PRINT# utasításnál.

A PRINT# és a CMD utasítás kapcsolata

A PRINT# utasítás feloldja a CMD hatását, a listakép kiértékelése után visszaállítja a képernyő hozzárendelését standard output készüléknek. (CLRCH KERNAL rutin) Ha a CMD utasítást program listázásához használjuk (LIST utasítás előtt), a listázás végrehajtása után a lista file-t csak PRINT# utasítást követően zárhatjuk le. (A CLOSE nem törli az output készülék kijelölését.) Ha a listázást követően erről megfeledkezünk, a következő I/O utasítás hatása bizonytalan.

Programhiba és a CMD utasítás kapcsolata

Bármely hibaállapot bekövetkezése feloldja a CMD hatását, mert a hibakezelést a BASIC interpreter "meleg start" rutinja végzi és ez tartalmazza a CLRCH KERNAL rutint.

Egy karakter nyomtatása

Egy listaelemmel kapcsolatos kiértékelés és végrehajtás egy karaktersorozat átvitelét jelenti az aktuális output készülékre, karakterenként. (A PRINT utasításnál alapállapotban ez a készülék a képernyő. Az alapállapot megváltoztatása a CMD utasítással lehetséges.)

Ha az output készülék nem a képernyő, a karakter változtatás nélkül kerül:

- a soros buszra, vagy
- a kazetta pufferbe, vagy
- az RS232 pufferbe

az aktuális készüléknek megfelelően. Ha a pufferbe került és a puffer megtelt, akkor kiíródik.

Ha az output készülék a képernyő, a végrehajtás a karakter értékétől függ. Ha az érték:

a/ Kisebb mint 32 (vezérlő karakter)
Megtörténik az értéknek megfelelő vezérlés. (Lásd a képernyőről szóló fejezetet.)

b/ Nagyobb mint 31, de kisebb mint 128
Átalakitás képernyőkódra, ennek beírása a képernyő RAM-ba és az aktuális karakter szín beírása a szín RAM-ba.

c/ Nagyobb mint 127, de kisebb mint 160 (vezérlő karakter)
Megtörténik az értéknek megfelelő vezérlés. (Lásd a képernyőről szóló fejezetet.)

d/ Nagyobb mint 159
Kód = kód - 128 és lásd b/.

Karakter átvitel és a KERNAL kapcsolata

A karakter átvitelt a KERNAL CHROUT ($FFD2) rutinja végzi. A rutin a 3-as lapon keresztül átírható. (Egy JMP ($0326) utasításon keresztül érhető el.) Így lehetőség van az átvitel megváltoztatására, saját rutin beillesztésére. (Lásd "A KERNAL I/O struktúrája" fejezetet is.)

Alkalmazási példák:

1. Egy nyomtatványt kívánunk készíteni, melyre nyomtatás után több kézi beírás kerül. A kézi beírásoknál a karakter helyeket "." jelöli. A listakép szerkesztésénél (a program egyszerűsítése miatt) tabulációs és helyköz funkciót használunk. Ezért célszerű az átviendő karakterek vizsgálata és a helyköz karakter cseréje "." karakterre.

A feladat néhány utasításos ASSEMBLER rutinnal oldható meg:

```
        *=$c000
        lda #chan<    ;=#$0b
        ldy #chan>    ;=#$c0
        sta $0326
        sty $0327
        rts
        cmp #$20      ;szóköz
        beq chan      ; kurzor jobbra
        cmp #$1d
        bne ret
chan    lda #46       ;"."
ret     jmp $f1ca
```

A rutin hívása: SYS49152

2. A karakterkészlet konverziója más ASCII karakterkészletnek megfelelően.

Itt megadjuk a szükséges néhány ASSEMBLER utasítást:

```
          *=$c000
          lda #$0b
          ldy #$c0
          sta $0326
          sty $0327
          rts
          stx $97a6
          tax
          lda ujtab,x
          ldx $97
          jmp $f1ca
ujtab     .text   új karakterkészlet
```

Az új karakterkészlet BASIC utasításokkal is betölthető:

```
          for i=49170 to 49170+254
          read A$
          poke i,asc(a$)
          next
          data karakterkészlet
```

3. Lista file létrehozása disketten, tetszőleges sorhosszúsággal.

Itt csak a feladat nagyvonalú megoldását adjuk meg, a részletes kidolgozást az olvasóra bízzuk.

a/ A BSOUT rutin elé saját rutint illesztünk be (lásd az előző példákat).

b/ A saját rutinban számoljuk, hogy hányszor hívtuk meg a rutint. Ha a meghívások száma elérte a kívánt sorhosszt, egy "carriage return"-t visszünk át a BSOUT KERNAL rutinnal.

Az utasításoknál lehetséges hibaüzenetek

- "file not open" (KERNAL)

  Az lf logikai file-szám egy még nem megnyitott file-ra hivatkozik.

- "device not present" (KERNAL)

  Az lf logikai file-számmal megnyitott file-hoz rendelt output készülék nem áll rendelkezésre. (Ha előzőleg a file-t csatornaszám megadása nélkül nyitottuk meg és a soros buszra egynél több bekapcsolt készülék van felfűzve, csak a file-ra kiadott második CMD vagy PRINT# utasítás vált ki hibaüzenetet.

  (A file megnyitást közvetlenül követő PRINT# vagy CMD utasítás "nem veszi észre", hogy az output készülék nincs bekapcsolva.)

- "not output file" (KERNAL)

  A CMD vagy PRINT# utasításban használt logikai file megnyitásában a billentyűzet szerepelt készülékként.

- "illegal quantity" (BASIC)

  A CMD vagy PRINT# utasításban használt logikai file-szám nem esik az 1-255 intervallumba. Az SPC vagy TAB funkcióban megadott aritmetikai kifejezés értéke nem esik a 0-255 intervallumba.

- "type mismatch" (BASIC)

A listakép valamely elemében a numerikus és string típusú elemek keverednek. Pl. PRINT A$+128

- "syntax" (BASIC)

A szintaxis hiba (lásd az utasítások szintaxisa fejezet elején).

Pl. PRINT#4 A$
(a PRINT#4 után hiányzik a ",")

Pl. PRINT*;B$;128
(a PRINT utasítás után álló "*" karakter az előzőekben definiált listaelem típusok egyikeként sem értékelhető ki.)

Példák:

1.
```
10 OPEN4,4
20 FOR I=1 TO 4
30 A(I)=I
40 NEXT
50 PRINT#4,"                    T A B L A Z A T"
60 PRINT#4
70 PRINT#4,"    oszlop1","oszlop2","oszlop3","oszlop4"
80 PRINT#4
90 FOR I=1 TO 4
100 PRINT#4,"ertek"I":"A(I)SPC(5);
110 NEXT
RUN
```

```
                              T Á B L Á Z A T

 oszlop1              oszlop2                  oszlop3            oszlop4

érték 1 : 1          érték 2 : 2              érték 3 : 3          érték 4 : 4
```

2.
```
PRINTTAB(20)"FEJLEC"
                    FEJLEC
```

3.
```
PRINT"          szoveg1"TAB(5)"szoveg2"
szoveg1szoveg2
```

4.
```
PRINT"          szoveg1"SPC(5)"szoveg2"
szoveg1        szoveg2
```

5.
```
A=22:B=33:C=44
PRINTA;B;C
PRINTSTR$(A);STR$(B);STR$(C)

  22  33  44
  22 33 44
```

6.
```
PRINT"abc";:PRINT"def"
abcdef
PRINT"abc":PRINT"def"
abc
def
PRINT"abc";" "CHR$(157)"def"
PRINT"abcdef"
abcdef
abcdef
```

```
7.  PRINT 128;1.3;1111111111.123
    PRINT   1 2 8  ; 1 .3;1 1 1 1 1 1 1 1 1 1  .1 2 3
    PRINT(128)(1.3)(1111111111.123)
      128  1.3  1.11111111E+09
      128  1.3  1.11111111E+09
      128  1.3  1.11111111E+09
```

INPUT, INPUT#

Feladata: BASIC változókba olvas be értékeket egy input eszközről.

Szintaxis:

INPUT [ds;] v1 [,v2] ...

INPUT#lf,v1 [,v2] ...

ahol
ds - string konstans, az ún. dialóg string
lf - nyitott input logikai file száma, aritmetikai kifejezés értéke a 0-255 intervallumba kell essen
v1,v2... - tetszőleges típusú változók vagy tömbelemek

Az input eszköz a billentyűzet, a kazettás egység, az RS232 vonal, a képernyő, vagy a soros busz (lemezegység) lehet.

Képernyő input esetén a képernyő adattartalmát olvassa az aktuális kurzor pozíciótól.

Az INPUT a billentyűzetről olvas, az INPUT# a megadott logikai fileból (amely természetesen lehet a billentyűzetre is megnyitva).

I Pontosabban csak az a különbség, hogy az INPUT# kijelöli az input I
I egységet, az INPUT nem, a standard input egység pedig a billentyűzet,s I
I ez BASIC programból nem is változtatható meg. (Ill. megváltoztatására I
I nincs BASIC utasítás.) I

Az INPUT működése

a/ Kiírja a dialóg stringet (ha van), egy "?"-et és egy szóközt.

b/ Várja az input mező(k) feltöltését a billentyűzetről. A képernyőn villog a kurzor. A beütést RETURN-nel zárjuk le. Ezután a "?"-től a sor végéig a képernyősor tartalmát áttölti az ún. input pufferbe. (A $200-as címtől 88 byte). A pufferben a sor végét bináris nulla jelzi.

I Az interpreter a CHRIN KERNAL rutint hívja, amelytől 1 byte-ot kap I
I vissza. Ezt leteszi az input puffer következő byte-jába, s ez cikliku- I
I san ismétlődik, amíg a CHRIN $0D-t (carriage return-t) nem ad. I
I Egy "képernyősor" max. hossza 80 byte, tehát lehet, hogy a képer- I
I nyőn két sorban jelenik meg. I

c/ Kiír egy CHR$(13)-at (a képernyő következő sorának elejére pozicionál). Ha rögzítés után a képernyősorban nincs adat (az input puffer üres), az INPUT utasítás végrehajtása itt befejeződik.

Az üres input a következő program rutinként való felhasználásával szűrhető ki:

```
10 POKE 512,0
20 INPUT "Ü";X,Y,Z
30 IF PEEK(512)=0 THEN PRINT"URES INPUT": GOTO 50
40 PRINTX,Y,Z
50 FOR I=1 TO 500: X=X: NEXT I
60 GOTO 20
```

d/ Ha a sor nem üres, az input pufferből feltölti a megadott változó(ka)t.

e/ Ha az összes megadott változót feltöltötte, de még nem érkezett el a sor végéig, "EXTRA IGNORED" üzenet jelenik meg, de a program tovább fut.

f/ Ha még nem sikerült az összes változót feltölteni,
- mert a sornak már vége, de a változó lista még nem merült ki: kiír két kérdőjelet és egy szóközt, majd a hiányzó mezők rögzítését várja;
- mert numerikus változó töltésekor a mező értéke nem numerikus: "REDO FROM START" üzenet jelenik meg, és az utasítás végrehajtása elölről kezdődik.

Az INPUT# működése

a/ Az lf logikai file-t jelöli ki input egységnek.

b/ Beolvas a file-ból egy rekordot az input pufferbe. (Rekord vége: carriage return vagy file vége.) Az input puffer hossza 88 byte, ennél hosszabb rekordot INPUT#-szel nem tudunk olvasni, programhibát okoz. (E korlát feloldására a példák között találhatunk egy ASSEMBLER nyelven írt rutint.)

I Az olvasás itt is a CHRIN segítségével történik byte-onként. I
I A "file vége" vagy hibaállapot az ST-ből kérdezhető le. (File vége: I
I ST=64; hiba: ST<>0 és ST<>64.) I
I A file végén túli olvasás hatására az interpreter végtelen ciklusba I
I eshet. Sajnos ez bekövetkezik akkor is, amikor a file-nak még nincs I
I vége, de már csak carriage return(ok)-et tartalmaz. I

c/ Az input pufferből feltölti a megadott változó(ka)t. Ha a rekordnak vége, de még nem sikerült az összes változót feltölteni, beolvassa a következő rekordot. Ha viszont a rekordnak nincs még vége, de már a változólista összes eleme értéket kapott, a rekord maradék része nem érhető el. (A következő INPUT# a következő rekordot olvassa.)

d/ Visszaállítja az alapértelmezés szerinti input készüléket (billentyűzet).

A rekordmezők elválasztása

Külön foglalkozunk azzal, hogy a változók feltöltésekor milyen jelek határolják a rekordmezőket. Az alábbiak egyaránt vonatkoznak az INPUT-ra és az INPUT#-re.

- A bináris nulla az egész rekordot terminálja, a rekord maradék része az INPUT# számára hozzáférhetetlen.
- A carriege return minden esetben terminál. (Ha a file-ban idézőjelek között van, akkor is.)
- A szóköz sohasem terminál.
  Numerikus változó feltöltésekor a mezőben lévő összes szóköz figyelmen kívül marad.
  String változó töltésekor a mezőt bevezető szóközök értéktelenek, a közbenső és záró szóközök azonban bekerülnek a változóba. (Kivéve,ha az input a billentyűzet, mert a sort (nem a mezőt) záró szóközöket a CHRIN levágja, s így azok nem kerülnek a pufferbe.)

Ha a fentieket összevetjük azzal, hogy a PRINT# numerikus érték után terminátorként szóközt ír ki, akkor egy kellemetlen jelenséget figyelhetünk meg, amelyet az alábbi példa szemléltet:

```
10 OPEN 7,8,9,"O:HOPPA,W"
20 A=1: B=2: C=3
30 PRINT#7,A;B;C
40 CLOSE 7
50 OPEN 7,8,9,"O:HOPPA"
60 INPUT#7,A,B,C
70 PRINT A;B;C
```

A képernyőn ez jelenik meg: 123 0 0
A kiírt számok tehát visszaolvasva "egybefolynak".
Oka: interpreter hiba.

- A vessző és a kettőspont numerikus és string mezőket egyaránt terminál. Ha egy string változóba olyan értéket akarunk beolvasni, amely vesszőt vagy kettőspontot tartalmaz, kiíráskor a mezőt idézőjelek közé kell tenni. Az interpreter két idézőjel vagy egy idézőjel és rekord/file vége közötti részt változatlanul vesz át.

Példák, megjegyzések:

1. Numerikus változók (X,Y,Z) értékének kiírása olyan formában, amelyet INPUT#-szel vissza is tudunk olvasni:

a/
```
10 PRINT#7,X
20 PRINT#7,Y
30 PRINT#7,Z
```

b/
```
10 T$=","
20 PRINT#7,X;T$;Y;T$;Z
```
A ";" itt lényeges!

c/
```
10 T$=","
20 PRINT#7,STR$(X)T$STR$(Y)T$STR$(Z)
```

A nehézkesnek tűnő c/ megoldás kevesebb helyet foglal az adathordozón (vo. PRINT# ut.)

2. Kurzor pozícionálása billentyűzet vagy képernyő input előtt:

```
100 POKE 214,LN         :REM SOR
105 POKE 211,CL         :REM OSZLOP
110 SYS58732
115 INPUT"SZAMLASZAM";SZ
```

I Billentyűzet input esetén a kurzort ugyanúgy bárhová elpozicionál- I
I hatjuk, mint parancsok vagy programszöveg begépelésekor. Itt azonban I
I inkább korlát, mint lehetőség. A következő példa ezt mutatja meg. I

3. 10 INPUT"HOL ERED A DUNA";A$

Ha most azt ütjük be, hogy "A FEKETE ERDOBEN" RETURN, akkor A$="A FEKETE ERDOBEN"

De ha azt ütöm be, hogy "kurzor le" "kurzor fel" A FEHER ERDOBEN "return", akkor

A$="HOL ERED A DUNA? A FEHER ERDOBEN"

I Ha rögzítés közben elpozicionáljuk a kurzort arról a sorról, ahol I
I az az INPUT utasítás belépésekor állt, akkor azt a sort fogja behozni I
I az input pufferbe, amelyen a kurzor áll akkor, amikor a RETURN billen- I
I tyűt lenyomjuk. A sort mindenképp elölről hozza be, akkor is, ha köz- I
I ben visszatértünk a kiinduló sorra. Ez megkérdőjelezi az INPUT utasí- I
I tás használhatóságát igényesebb adatrögzítő programokban. I

4. Az INPUT utasításban elkövetett szintaktikai hibát az interpreter nem mindig ismeri fel:

10 INPUT!<>=

Kéri az adatot, mielőtt észre venné a hibát. Ha nem ütünk be semmit,az utasítás végrehajtása befejeződik.

Példák:

a/ A C 64 nem az INPUT ill. INPUT# utasításoknak köszönheti népszerűségét. Az alábbi rutin az INPUT# korlátait oldja fel. Adott számú byte-ot olvas egy már megnyitott logikai file-ből egy string változóba.

A rutin hívása: SYS49152,lf,hossz,stringváltozó

Pl. SYS49152,4,137,A$(I)

```
*=49152
JSR $AEFD           ; "," vizsgálata
JSR $B79E           ; file-szám
STX $13
```

```
        JSR $AEFD           ; "," vizsgálata
        JSR $B79E           ; input hossza
        STX $61
        JSR $AEFD           ; "," vizsgálata
        JSR $B08B           ; a stringváltozó megkeresése
        LDA $61
        BNE NONULL
        JMP $ABBA
NONULL  JSR $B6A3
        LDA $61
        JSR $B475           ; helyfoglalás a stringnek
        LDY #2
LB01    LDA $61,Y
        STA ($47),Y
        DEY
        BPL LB01
        LDX $13
        JSR $E11E           ; CHKIN
        LDY #0
LB02    JSR $FFCF           ; input egy byte
        STA ($62),Y
        INY
        CPY $61
        BNE LB02
        JMP $ABB5           ; input készülék visszaállítása
```

b/ Legyen egy (szekvenciális) file adattartalma a következő:

```
12 3 4,25:"NEV:NAGY",EV:85 36.45

                 CHR$(13) CHR$(13)
                          file vége
```

A `10 INPUT#LF,A, B, C$, D$, E, F`

utasítás végrehajtása után a változók:

```
A=1234
B=25
C$=NEV:NAGY
D$=EV
E=85
F=36.45
```

Lehetséges hibák:

- ILLEGAL QUANTITY — ha lf<0 vagy lf>255
- FILE NOT OPEN — ha az lf file nincs megnyitva

- STRING TOO LONG	ha a rekord nem fér be az input pufferbe (rekordhossz>88)
- FILE DATA	csak INPUT#-nél: a program numerikus értéket vár, de a file-ből nem azt kap
- REDO FROM START	csak INPUT-nál: a program numerikus értéket vár, de nem azt gépeltünk be
- DEVICE NOT PRESENT	a készülék nem létezik vagy nincs bekapcsolva
- ILLEGAL DIRECT	parancsmódban próbáltuk kiadni az INPUT, ill. INPUT# utasítást
- NOT INPUT FILE	ha a készülék a kazetta, és a másodlagos cím 0

GET

Feladata: A billentyűzetről olvas egy-egy byte-ot a megadott változó(k)ba.

Szintaxis:

GET s1$ [,s2$] ...

ahol sn$ string változó vagy string tömb elem.

I Elvben megengedett string változó helyett numerikus változó használata is. Ekkor csak a 0-9 billentyűket fogadja el, s a megfelelő számértéket tölti a numerikus változóba. Ez az opció a gyakorlatban használhatatlan, mert a 0-9-től eltérő billentyű leütése esetén a program SYNTAX ERROR-ral megszakad. I

Csak programban működik, parancsként nem.

Nem várakozik: ha nincs leütés, a string változó értéke nulla hosszú (üres) string lesz.

A kurzor nem villog.

I Valójában nem a GET olvas, hanem a megszakítás rutin, amely 20 ms-onként kapja meg a vezérlést és - többek között - lekérdezi és dekódolja a billentyűzetet, majd a leütött billentyű kódját elteszi a billentyűzet pufferbe (címe $277, hossza 10 byte; a $s6 címen található, hogy éppen hány byte van benne), a GET onnan veszi elő. I

A STOP leütése megszakítja a programot. (Nem úgy, mint az INPUT utasítás esetén.)

Példák:

a/
```
15 GET A$
20 IF A$="" THEN 15
```

Vár egy billentyű leütésére. A leütött billentyű az A$-ban.

b/
```
1000 GET KY$
1005 IF KY$<>CHR$(13) AND KY$<>CHR$(147) THEN 1000
```

Vár, amíg a RETURN vagy a SHIFT-elt CLR/HOME le nincs ütve. A billentyű kódja a KY$-ban.

c/
```
100 GET FU$
110 IF FU$<CHR$(133) OR FU$>CHR$(140) OR FU$="" THEN GOTO 100
```

Egy funkció billentyű (f1-f8) leütését várja.

d/
```
45 POKE 198,0:WAIT 198,1
50 GET A$
```

A 45-ös sor törli a billentyűzet puffert és vár, amíg az üres.

e/
```
120 GET A$,B$
130 IF A$<>"" AND B$="" THEN 1200
```

Ha egy, de csak egy billentyű volt leütve, az 1200-as sorra ugrik.

Lehetséges hibák:

- SYNTAX ERROR
- ILLEGAL DIRECT

GET#

Feladata: Az lf logikai file-ból olvas be egy-egy byte-ot a megadott változó(k)ba.

Szintaxis:

```
GET#lf,s1$ [,s2$] ...
```

ahol lf egy nyitott input file száma, aritmetikai kifejezés,
sn$ string változó vagy string tömb elem.

Csak program módban működik.

Ha a logikai file-t a billentyűzetre nyitottuk meg, a GET# működése azonos a GET-ével, ld. ott.

Az input készülék a billentyűzeten kívül lehet a képernyő, a kazettás egység, a floppy, vagy az RS232 vonal.

A képernyőről az aktuális kurzor pozíción lévő karaktert olvassa be, a kazettás egységről, vagy a floppyról az adott file következő byte-ját. Az utóbbi esetben a "file vége" vagy a hibaállapot az ST státusz változóból lekérdezhető. (File vége: ST=64, hiba:ST<>0 és ST<>64,értelmezése berendezésfüggő.) A "file vége" állapot a file utolsó byte-jának (tehát nem az utolsó utáninak) olvasásakor következik be.

Ha a beolvasott byte bináris nulla, a változóba nem CHR$(0),hanem üres string kerül.

A GET# működése

- Kijelöli az input egységet (CHKIN KERNAL rutin, vo. CMD, PRINT#, INPUT#; ld. meg A KERNAL I/O struktúrája).
- Beolvassa a byte-o(ka)t.
- Törli az input kijelölést (visszaállítja az alapértelmezést).

Példák:

a/ Egyszerű (de lassú) file másoló. A 8-as floppy egységről másol át egy program file-t a 9-esre:

```
5  INPUT "FILE";FL$
10 OPEN 1,8,7,"0:"+FL$
15 IF ST THEN THEN 50
20 OPEN 2,9,7,"@0:"+FL$+"P,W"
25 GET#1,B$:IF ST THEN 50
30 PRINT#2,LEFT$(B$+CHR$(0),1);
35 GOTO 25
50 IF ST<>64 THEN PRINT"HIBA"
55 PRINT#2,LEFT$(B$+CHR$(0),1);
60 CLOSE 1:CLOSE 2:END
```

Az 5-ös sorban kéri a program nevét, majd megnyitja az input és az output file-t.

A 30-as sorban azért nem megy a sima PRINT#2,B$, mert bináris nullák átvitele esetén B$="", s így azok nem kerülnének át az outputra. Az 55-ös sor megegyezik a 30-assal, itt íródik ki az utolsó byte.

Lehetséges hibaüzenetek:

ILLEGAL QUANTITY	lf<0 vagy >255
FILE NOT OPEN	az adott logikai file nincs megnyitva
DEVICE NOT PRESENT	a file-hoz rendelt készülék nem érhető el, pl. nincs bekapcsolva vagy nem is létezik
ILLEGAL DIRECT	az utasítás parancs (direkt) módban nem használható

LOAD

Feladata: Programfile-t tölt a memóriába

Szintaxis:

LOAD [fnév] [,ks] [,tk]

ahol fnév a programfile neve, string kifejezés
ks készülékszám, aritmetikai kifejezés, alapértelmezése: 1
tk töltési kód, aritmetikai kifejezés, alapértelmezése: 0

A készülék a kazettás egység (1) vagy floppy meghajtó (8-11) lehet. Utóbbi esetben kötelező a file-név megadása. Az RS232 vonalról és a billentyűzetről nem lehet LOAD-olni.

A harmadik paraméternek (töltési kód) kazetta input esetén nincs hatása. Ha a megadott készülék a floppy, akkor a töltési kód értékétől függ, hogy a LOAD hova tölti a programot. Ha ez az érték 0, akkor a BASIC terület elejére, ha nem 0, akkor az eredeti helyére (ahonnan ki lett mentve). Utóbbi lehetőséget elsősorban gépi kódú programok töltésekor használjuk.

A BASIC terület kezdőcímét a 43-44-es ($2b/2c) címen található rendszerváltozó értéke adja, a program eredeti kezdőcímét pedig a programfile első két byte-ja.

A lemezes file-ok azonositására a file-névben speciális karakter, az ún. JOKER is használható. Két JOKER van, a "?" és a "*".

Ha a file-név végén "*" áll, akkor ez a diszkette tartalomjegyzékéből azt a programnevet választja ki, amely az adott file-névvel kezdődik, vagy azzal azonos. Ha több ilyen van, akkor a lemezegység katalógusában először megtaláltat. Pl. a FO* file-névvel hivatkozhatunk a FO, FOKONYV, FOVAROS, FOMUFTI programokra.

A file-névben szereplő "?" tetszőleges karaktert jelöl. Pl. a ???? file-név az első 4 betűs file-névre hivatkozik (ha van ilyen), a ??F* pedig arra a file-ra, amelynek harmadik betűje F.

A SHIFT-tel együtt leütött RUN billentyű egy paraméter nélküli LOAD utasítást generál, pontosabban hatása a

```
LOAD return
RUN return
```

parancsok beütésével azonos. A paraméter nélküli LOAD a kazettás egységről olvassa be a következő file-t. Ha a programot azonnal elindítani is akarjuk:

```
LOAD "név",8,0 és (SHIFT/stop)
```

A LOAD parancs másképp működik, mint a LOAD programutasítás. (Az egyetlen C 64 BASIC utasítás, amelyre ez igaz, ha eltekintünk attól, hogy néhány utasítás (pl. INPUT,GET) egyáltalán nem működik parancs módban.)

A LOAD utasítás működése parancs módban:

- Kiírja a képernyőre, h. SEARCHING FOR fnév (floppy), vagy PRESS PLAY ON TAPE (kazetta)
- Megnyitja a programfile-t.
- Ha ez sikerült, kiírja, h. LOADING (floppy), vagy FOUND fnév (kazetta).
- Betölti a file-t. A töltés végén a program utáni első szabad byte címe a "betöltött program vége" pointerben rendelkezésre áll. (174-175-ös címen)
  Ezen a címen kezdődik a változóterület (a 45-46 ($2d-$2e) rendszer változó mutat a változóterület elejére).
- Lezárja a programfile-t.
- Kiad egy CLR utasítást.
- Relokálja a BASIC területen található program "következő sor" pointereit.

A "következő so." pointerekről ld. a BASIC sor felépítését a LIST utasításnál.

Ezek relokálása biztositja, hogy a BASIC terület a memóriában áthelyezhető, a BASIC programok bárhol futtathatók.

A töltés folyamata STOP billentyűvel megszakítható.

Példák:

a/ LOAD "KUKAC",8: PRINT "INDUL":RUN

Betölti a KUKAC programot a 8-as floppy 0-as meghajtójáról, de nem írja ki, hogy INDUL és nem is indítja el, mert a LOAD után új parancsot vár és a sor maradék részét "lenyeli".

b/ ALMA egy gépi kódú program, de a BASIC területén szeretnénk futtatni:

```
LOAD "ALMA",9
SYS 2049          :REM A BASIC TERÜLET ELEJE
```

Ez nem megy így, mert a "következő sor" pointereket a BASIC a gépi kódú programunkban fogja elhelyezni és tönkre teszi azt. Csak úgy védhető ki, hogy a gépi kódú program elejére két bináris 0-at teszünk. (BASIC program vége jelzés.)

c/

```
LOAD"PRINT£",8,1
SYS828            :REM KAZETTAPUFFER CÍME
```

A PRINT£ gépi kódú programot tölti és inditja, feltéve természetesen, hogy töltési és indulási címe egyaránt 828.

d/ LOAD"D*",8

A 8-as floppyról tölti be az első D betűvel kezdődő programot.

e/ LOAD"$",9

A 9-es floppy egységre feltett diszkett tartalomjegyzékét tölti a BASIC területre. (Az esetleg ott lévő program elvész!) A tartalomjegyzék LIST-tel jeleníthető meg.

A LOAD utasítás működése program módban

- Megnyitja a megadott programfile-t (kazetta esetén még kiírja: PRESS PLAY ON TAPE).
- Ha ez sikerült, betölti a programot és aktualizálja a "betöltött program vége" pointert.
- Relokálja a "következő sor" pointereket a BASIC területen.
- Végrehajt egy RESTORE utasitást.
- Törli a vermet (a FOR és GOSUB bejegyzéseket).
- Elindítja a BASIC területen lévő programot.

Tehát program módban nem aktualizálja a "változók kezdete" pointert és nem ad ki CLR-t. Ez lehetőséget ad arra, hogy a betöltött program használja a töltőprogram változóit, viszont kizárja azt, hogy a program egy nálánál hosszabb programot magára töltsön.

A string változók közül azok, amelyek string konstanssal kaptak értéket, a LOAD hatására elvesznek. (Hosszuk marad a régi, tartalmuk azonban indefinit.) Ha szükségünk van a változók tartalmára, a programban úgy kell értéket adnunk nekik, hogy a stringterületre kerüljenek. (Pl. A$=""+"CONSTANS") Elvesznek továbbá a definiált függvények is (DEF FN).

Ha a betöltött program hosszabb, mint amelyik betöltötte, akkor a betöltött program elején aktualizálni kell a "változók kezdete" pointert és ki kell adni egy CLR-t:

Példák:

f/ Gépi kódú rutin töltése programból

```
1  LOAD"PRINTUSING",8,1
```

Ez így végtelen ciklus. (A LOAD után a program újra indul.)
Helyesen pl. így:

```
1 IF LD THEN 3
2 LD=-1:LOAD"PRINTUSING",8,1
3 ...
```

vagy így:

```
1 IF PEEK(512)=0 THEN 3
2 POKE 512,0:LOAD"PRINTUSING",8,1
3 ...
```

Kazetta esetén csak ez az eljárás lehetséges.

Lehetséges hibaüzenetek:

TYPE MISMATCH	pl. a file-név nem string típusú
MISSING FILE NAME	a készülék floppy, de a file-név hiányzik v. üres string
ILLEGAL QUANTITY	a készülékszám, vagy a töltési kód nem esik a 0-255 intervallumba
ILLEGAL DEVICE NUMBER	a készülékszám 0, 2, vagy 3
DEVICE NOT PRESENT	a megadott floppy nincs bekapcsolva

SAVE

Szintaxis:

SAVE [fnév] [,ks] [,tk]

ahol fnév a programfile neve, string kifejezés

ks készülékszám, aritmetikai kifejezés, alapértelmezése: 1

tk aritmetikai kifejezés. Az utasítás végrehajtására semmilyen hatása nincs.

A készülék a kazettás egység (1) vagy floppy meghajtó (8-11) lehet. Utóbbi esetben kötelező a file-név megadása. Az RS232 vonalra nem lehet programot menteni. A SAVE a BASIC terület elejétől a változóterületig ment, tehát a változók nem kerülnek kiírásra.

Az első kimentett byte címe: ($2B-$2C)

Az utolsó byte címe: ($2D-$2E)-1

Működése:

- Üzenet a képernyőn: PRESS RECORD & PLAY ON TAPE (csak kazettánál) és SAVING fnév
- Megnyitja a programfile-t.
- Kiírja a memória tartalmát.
- Lezárja a file-t.

A SAVE nem "veszi észre", ha a programfile-t nem sikerült megnyitni, ezért SAVE után ajánlatos VERIFY-t is kiadni.

Példák:

a/ SAVE "EDITOR",8

Ha most a diszketten van EDITOR nevű file,a mentés nem hajtódik végre.

b/ SAVE "@:EDITOR",8

A "@" nem része a file-névnek. Segítségével arra utasítjuk a floppy vezérlőt, hogy ha a diszketten már van EDITOR nevű program file, akkor írja felül azt.

Lehetséges hibaüzenetek:

TYPE MISMATH	ha az első paraméter nem string, vagy a továbbiak nem numerikusak
MISSING FILE NAME	ha a készülék lemezegység és hiányzik a file-név
ILLEGAL DEVICE NUMBER	ha a készülék 0, 2, vagy 3
ILLEGAL QUANTITY	ha a készülékszám < 0 vagy > 255
DEVICE NOT PRESENT	ha a készülék nem létezik vagy nincs bekapcsolva

VERIFY

Feladata: Programfile-t hasonlít össze a memória tartalmával.

Szintaxis:

VERIFY [fnév] [,ks] [,tk]

ahol fnév a programfile neve, string kifejezés

ks készülékszám, aritmetikai kifejezés, alapértelmezése: 1

tk töltési kód, alapértelmezése: 0

A készülék a kazettás egység (1) vagy floppy meghajtó (8-12) lehet. Utóbbi esetben kötelező a file-név megadása.

A harmadik paraméternek (töltési kód) kazetta input esetén nincs hatása. Ha a megadott készülék a floppy, akkor a töltési kód értékétől függ, hogy a VERIFY a memória melyik részével hasonlítja a programot. Ha ez az érték 0, akkor a BASIC területtel, ha nem 0, akkor azzal a területtel, ahonnan a program ki lett mentve. (A program file első két byte-ja tartalmazza a címet.)

Lemezes file-ok azonosítására a file-névben speciális karakter, az ún. JOKER is használható. (Ld. LOAD utasítás)

Működése:

- Kiírja a képernyőre, hogy SEARCHING FOR file-név (floppy),vagy PRESS PLAY ON TAPE (kazetta).
- Megnyitja a programfile-t.
- Ha ez sikerült, kiírja, hogy VERIFYING (floppy), vagy FOUND file-név (kazetta).
- Elvégzi az ellenőrzést.
- Lezárja a file-t.

Példák:

a/ SAVE"PROG1",8
VERIFY"PROG1",8

PROG1 néven kimenti a BASIC területen lévő programot és ellenőrzi a mentés helyességét.

b/ VERIFY"PRINTUSING",8,1

A PRINTUSING gépi kódú programot hasonlítja a memóriában lévő változathoz.

Lehetséges hibaüzenetek:

TYPE MISMATCH	pl. a file-név nem string típusú
MISSING FILE NAME	a készülék floppy, de a file-név hiányzik, vagy üres string

ILLEGAL DEVICE NUMBER	a készülékszám 0, 2, vagy 3
DEVICE NOT PRESENT	a megadott floppy nincs bekapcsolva vagy nem létezik
FILE NOT FOUND	a keresett programfile-t nem találja

LIST

Feladata: program listázása

Szintaxis: LIST [s1] [-s2]

s1 programsor száma, amelytől a program listázandó

s2 programsor száma, ameddig a program listázandó

Ha az s1 paraméter nincs megadva, értéke alapértelmezésként: 0
Ha az s2 paraméter nincs megadva, értéke alapértelmezésként: 65535

Az utasítás végrehajtása parancs módban

A BASIC interpreter kiértékeli a paramétereket. Ha valamelyik paraméter hiányzik, helyettesíti az alapértelmezés szerinti értékkel.

Az s1 által definiált sor keresése az aktuális BASIC RAM elejétől (a $2B/$2C rendszerváltozó mutat a címre) kezdődik. Ha az s1. számú sor a programban nem létezik, a listázás az ezt követő legkisebb számú sortól kezdődik. Az utolsó listázandó sort az s2 paraméter határozza meg. Ha az s2. számú sor a programban nem létezik, a listázás az s2.-at megelőző legnagyobb számú sornál ér véget. A listázás a programszöveg soronkénti dekodolását és átvitelét jelenti az aktuális output készülékre, mely alapállapotban a képernyő. (CMD utasítással változtatható meg.)

A programszöveg végét az interpreter számára az utolsó sor után lévő két bináris nulla jelenti.

A listázást tehát a változók kezdetét mutató rendszerváltozó ($2D/2F) értéke nem terminálja.

Nem terminálja a listázást a BASIC RAM végét jelentő rendszerváltozó ($37/$38) értéke sem.

A program dekódolása

Bár jelen könyv témájához ez szorosan nem tartozik, kitérünk a programsor felépítésére:

```
|..|..||..|..||  kódolt BASIC sor |0|  .....  |0 ||00
  ↑       ↑                        ↑            ↑
következő a sor                   sor         program
sor címe  száma                   végjel      végjel
```

A sor dekódolása és a dekódolt szöveg átvitele az output készülékre karakterenként történik. Az átvitel a dekódolt sorszám átvitelével kezdődik, ezt egy szóköz átvitele követi.

A BASIC sor dekódolásában a ("") karakter kiemelt jelentőségű. Két ("") karakter közötti karaktersorozat dekódolás nélkül kerül átvitelre. Szintén nincs dekódolás, ha a karakter értéke kisebb, mint 128.

A dekódolás lényegében a tokenizálás ellentétje. Dekódolás után a megfelelő utasítás vagy funkciónév karakterenként átvitelre kerül. Egy karakter átvitele a PRINT utasításnál leírtakkal azonos.

Az utasítás végrehajtása program módban

A végrehajtás a fentiekkel azonos, de a végrehajtás után a program megszakad és CONT utasítással sem folytatható. (A végrehajtás után - függetlenül a program vagy direkt módtól - a BASIC interpreter "meleg start" rutinja működik.)

Megjegyzések:

a/ Az interpreter a REM utasítás utáni karaktersorozatot is dekodolja.

b/ Ha REM utasítás után a dekodolandó karakter ASCII értéke 204 vagy 108, a listázás "syntax error" üzenettel megszakad. (Interpreter hiba.)

c/ Könnyen elkövethető és nehezen "megfejthető" programozási hiba a PRINT# utasítás begépelésénél a "?#" rövidített megadás. A tokenizált, majd dekodolt forma azonos a PRINT# tokenjének dekodolt formájával, a végrehajtás alatt álló program azonban "syntax error" üzenettel megszakad.

d/ A paraméterek megadásánál elkövetett szintaxis hiba esetén, ha a hiba a "-" jel és a második paraméter között van, hibajelzés nincs és a második paraméter értéke alapértelmezés szerinti lesz (65535). Pl. LIST 10 - *20 a listázás a program végéig történik.

"Beavatkozás a dekódolásba

A dekódolást 3-as lapon keresztül elérhető rutin végzi. Így lehetőség van a dekódolás megváltoztatására, saját rutin beillesztésére.

Pl. Ha úgy definiáltunk új BASIC utasítást, hogy az utasítás (a tokenizáló rutin átírásával) új tokent kapott, a tokent itt kell dekódolnunk és az utasítás szövegét karakterenként átvinnünk az aktuális output készülékre.

Új utasításunk neve legyen NYOMTATÁS, az új token legyen a 205 ASCII értékű karakter.

A megoldást assemblerben adjuk meg.

```
*=$c000
lda #$0b
ldy #$c0
sta $0306
sty $0307
rts
cmp #205
beq list
```

```
        jmp $a71a
list    ldx #$00
listl   lda const,x
        cmp #$ff
        beq ret
        jsr $ab47
        inx
        jmp listl
ret     jmp $a6f6
const   .text
        .byte $ff
```

I Az assembler rutin BASIC-ból a SYS49512 utasítással aktivizálható. I
I A rutin mintájára írhatunk olyan rutint, amellyel a listázás ellen I
I "védett" program is listázható. (Szokásos védelem a BASIC szövegben I
I helyenként 204 vagy 108 értékű karakterek elhelyezése, lásd a megjegy- I
I zéseket.) I
I A könnyen listázhatóság érdekében célszerű rutinunkat kiegészíteni a I
I DEL karaktereket figyelő és helyettesítő utasítással is. I

Az utasításnál lehetséges hibaüzenet csak a "syntax".

Akkor kapjuk, ha az utasítás nem sorszámmal vagy "-" jellel kezdődik, illetve ha sorszámmal kezdődik, de nem "-" jellel folytatódik.

Példák az utasítás alkalmazására

LIST	A BASIC területen lévő programot (vagy bármit,ami ott van) kilistázza a BASIC terület elejétől, addig, amíg három zéró értékű byte-ot talál (ha nem BASIC program van a BASIC területen).
LIST100-	A BASIC területen lévő programot kilistázza a 100 sorszámú sortól a program végéig.
LIST-100	A programot a 100 sorszámú sorig listázza ki.
LIST10-100	A programot a 10 sorszámú sortól a 100 sorszámú sorig listázza ki.
LIST0-A	Hatása azonos a paraméter nélküli utasítással.

PERIFÉRIÁK, HARDWARE

KÉPERNYŐ

A képernyőt - a könyv témájához kapcsolódva - csak röviden, az adatfeldolgozási I/O szempontjából tárgyaljuk. Nem térünk ki részletesen a grafikai lehetőségekre, a sprite kezelésre.

Képernyőként TV-készülék vagy monitor használható. A C 64 összekapcsolása a képernyővel történhet:

- Koaxiális kábellel (TV vagy nagyfrekvenciás bemenettel is rendelkező monitor esetén). A csatlakoztatás a C 64 nagyfrekvenciás kimenete és a TV antenna bemenete között történik. A kapcsolatból adódóan a kép minősége - különösen gyengébb színes TV használata esetén - nem jó. A kapcsolat a szabványos 36-os UHF csatornán keresztül jöhet létre.
- Video kábellel (monitor, ill. video bemenettel rendelkező TV esetén). A csatlakoztatás a C 64 audio-video kimenete és a képernyő video bemenete között történik. A kép minősége jó, a karakterek kontrasztosak, "szellemképek" nincsenek. A megoldást hátráltatja, hogy még a video bemenettel rendelkező TV készülékekhez sem kapható video kábel, ill. az össze sem szerelhető (sem a C 64 aljzatába, sem a TV aljzatába illő dugó, sem a szükséges többpólusú árnyékolt kábel nem kapható).

Képernyő felépítése, szerkesztése

A képernyő kezelését a video-controller (VIC 6567) chip végzi. A chip 5 féle üzemmódban dolgozhat:

- karakter üzemmód
- többszínű karakter üzemmód
- változtatható háttérszín üzemmód
- bit térképes üzemmód
- többszínű bit térképes üzemmód.

A továbbiakban a karakter üzemmódot tárgyaljuk.

A képernyő felépítése:

- A C 64 a képernyőt 200*320 pontra bontja. Egy karakter ábrázolása 8*8 képponton történik, így a képernyő 25 karakter sorból és 40 oszlopból áll (számozásuk 0-24, ill. 0-39). Ez a felépítés a "LO-RES" (kisfelbontású) üzemmód. A gép két karakterkészletet használ. Az egyik a nagybetűk/grafikus jelek (grafikus mód), a másik a kisbetűk/nagybetűk (text mód). Mindkét karakterkészlet 128 jelből áll. Minden jelnek létezik az inverze.
- A képernyő memória (alap állapotban) az 1024 - 2023 tárcímeken, a szín memória az 55296 - 56295 címeken található. A két karakterkészlet (a karakterek képe) ROM-ban van az 53248 - 56832 címig. A képernyő memória (más gépektől eltérően) a karakter kódját tárolja, a karakter megjelenítéséhez a VIC chip veszi ki a karakter képet (bit mintáját) a karakter ROM-ból, a kódnak megfelelően. A karakterkészlet címe RAM-ba is átcímezhető, így saját karakterkészlet is definiálható. A képernyő memória egy byte-jának értéke határozza meg a karakter képet, a byte helye a karakter helyét a képernyőn. A szín memóriában azonos relatív cím a karakter színét határozza meg. A képernyő tároló átcímezhető, a szín RAM nem. 16 szín állítható be (a színkódot csak a jobboldali félbyte tárolja).

- A képernyőre írni lehet:
  - valamelyik billentyű megnyomásával,
  - PRINT utasítással,
  - PRINT# utasítással (előtte a képernyőre egy file-t kell megnyitnunk),
  - POKE segítségével: a képernyő memóriába betesszük a megfelelő byte-ba a karakter kódját, a szín memóriába beírjuk a szín kódját.
- A képernyő két részből áll, a 25*40 karakternyi funkcionális belső részből és a keretből. Mindkettő alapszíne külön állítható, itt is 16 szín használható.

A képernyőt kezelő funkció-billentyűk ill. utasítások:

- Háttér és a keret színe csak utasítással állítható:
  POKE53280,s ill.
  POKE53281,s (s: a szín kódja)
  0 - fekete
  1 - fehér
  2 - piros
  3 - türkisz
  4 - bíbor
  5 - szürke
  6 - kék
  7 - sárga
  8 - narancs
  9 - barna
  10 - világos piros
  11 - világos szürke
  12 - közép szürke
  13 - világos zöld
  14 - világos kék
  15 - sötét szürke
- A karakterek színe beállítható a CTRL ill. COMMODORE billentyű és a megfelelő színbillentyű (1-8) együttes megnyomásával.

Beállítható a karakter szín PRINT utasítással.
PRINT"s", s: a képernyőn megjelenő grafikus jel, melyet CTRL és a színbillentyű együttes megnyomása ad PRINTCHR$(n), n: színnek megfelelő ASCII érték:

5 - fehér
28 - piros
30 - zöld
31 - kék
129 - narancs
149 - barna
150 - rózsaszín
151 - sötét szürke
152 - közép szürke
153 - világos zöld
154 - világos kék
155 - világos szürke
156 - bíbor
158 - sárga
159 - cinober

Beállítható egy karakter színe POKE utasítással, a szín memória megfelelő byte-jának állításával.

- Inverz be- ill. kikapcsolás lehetséges a CTRL és az RVS ON ill. RVS OFF billentyűk együttes megnyomásával.

PRINT utasítással

PRINT"r", r: a képernyőn megjelenő grafikus jel, melyet a CTRL és RVS-billentyű együttes megnyomása ad PRINTCHR$(k) k: a vezérlésnek megfelelő ASCII érték

18 - inverz mód bekapcsolás
146 - inverz mód kikapcsolás

- A kurzor mozgatása és a karakter törlés/beszúrás a megfelelő funkció-billentyű megnyomásával, ill. PRINT utasítással történhet.

PRINT"k", k: a képernyőn megjelenő grafikus jel, melyet a funkció-billentyű megnyomása ad "idézőjel módban" (erre később visszatérünk).

PRINTCHR$(k), k: a mozgatásnak megfelelő ASCII érték

29 - kurzor jobbra
157 - kurzor balra
145 - kurzor fel
17 - kurzor le
19 - kurzor a bal felső sarokba
147 - képernyő törlés és kurzor a bal felső sarokba
148 - karakter beszúrás
20 - karakter törlés.

- Karakterkészlet kijelölés lehetséges a SHIFT és a COMMODORE billentyű együttes megnyomásával, PRINT utasítással, ill. POKE segítségével.

PRINTCHR$(k), k: ASCII érték

14 - átváltás kisbetűk/nagybetűk (text) módra
142 - átváltás nagybetűk/grafikus jelek (grafikus) módra

POKE53272,k

23 - átváltás kisbetűk/nagybetűk (text) módra
21 - átváltás nagybetűk/grafikus jelek (grafikus) módra.

- Karakterkészlet átváltásának megtiltása print utasítással
  - PRINTCHR$(8): átváltás letiltása
  - PRINTCHR$(9): letiltás feloldása

Képernyő kód - ASCII kód

Előzőleg említettük, hogy a képernyőre lehet írni PRINT utasítással, ill. a képernyő memóriába töltéssel. A képernyő kód és a PRINT utasításban használt ASCII kód [CHR$(k)] nem azonos. Ha PRINT-et használunk, az utasítás alakítja át az ASCII kódot a képernyő memóriába töltés előtt. Ha az íráshoz a memóriába töltést használjuk, gondoskodnunk kell a kód konverzióról. Pl. a POKE1024,ASC("X$") utasítás eredménye a képernyő bal sarkában megjelenő karakter. Ez a karakter csak akkor egyzik meg X$ első jelével, ha a két kód véletlenül azonos. A számok, betűk, írásjelek, operátorok kódja azonos.

A kódtáblázatot a képernyő kódok szerint adjuk meg az A függelékben. A karaktereknél az 1. oszlop a grafikus módra, a 2. oszlop a text módra vonatkozik. Az ASCII kódoknál azt a kódot adtuk meg, ami a PRINTCHR$(kód)utasításban a képernyőn az adott karaktert jeleníti meg.

Kurzor pozicionálása

Ha a képernyőre PRINT utasítással írunk, a kiírt szöveg ott fog kezdődni, ahol a kurzor áll. Adott sor, oszlop pozíciótól íráshoz a kurzort a megadott helyre kell vinni az írás előtt. A memóriába közvetlenül írva (pl. POKE), gondoskodnunk kell írás előtt a kód konverzióról. Mindkét módszer nehézkes. Nézzünk példákat:

1. Megjelenítendő a képernyő 10. sorának 10. oszlopától az "12345678" szöveg

```
10 PRINTCHR$(19);: REM kurzor a bal felső sarokba
20 FORI=1 TO10:PRINTCHR$(17);:NEXT: REM sor pozicionálás
30 FORI=1 TO 9:PRINTCHR$(29);:NEXT: REM oszlop poz.
40 PRINT"12345678"
```

A módszer játéknak jó, de képernyő kezelő programnál elfogadhatatlan.

2. A feladat legyen az előbbi, de most próbáljuk meg a másik módszert

```
10 S=10: REM sor beállítás
20 O=9 : REM oszlop beállítás
30 A$="12345678"
40 M1=1024+S*40+O: REM képernyő memória
50 M2=55296+S*40+O: REM szín memória
60 FOR I=1 TO LEN(A$)
70 POKEM1+I,ASC(MID$(A$,I,1))
80 POKEM2+I,1: REM fehér
90 NEXT
```

Mint látjuk ez a módszer sem túl egyszerű, annak ellenére, hogy a kód konverziót kikerültük (a kiírandó string karaktereinek képernyő és ASCII kódja azonos).

3. A feladat újra az előbbi, de egy valamivel használhatóbb módszer:

```
10 FOR I=1 TO 24: LE$=LE$+CHR$(17): NEXT
20     REM KURZOR LE
30 FOR I=1 TO 39: JOBB$=JOBB$+CHR$(29):NEXT
40     REM KURZOR JOBBRA
50 HOME$=CHR$(19)
60     REM KURZOR A BAL FELSŐ SAROKBA
70 PRINT HOME$;LEFT$(LE$,10);LEFT$(JOBB$,9);"12345678"
```

Tehát hiányzik a más gépeknél megszokott PRINT AT utasítás. Az adatfeldolgozásnál használatos BASIC bővítő interpreterek tartalmaznak valamilyen pozícionáló utasítást. Itt egy, a standard interpreter rutinját hívó megoldást adunk:

```
POKE 214,S: REM sor
POKE 211,O: REM oszlop
SYS 58640 : REM kurzor beültetés
```

A képernyő sorvezérlése

Említettük, hogy a képernyő 25, 40 karakteres sorból áll. Az interpreter számára egy sor max. 80 karakter lehet. Ezt az ellentmondást a sorvezérlés oldja fel. A képernyő 1-24 számú sora folytatása lehet az előző sornak. Így a sorméret 80 karakter lehet. A sor méretét a képernyőre írás állítja be, általában 80 karakterre (dupla sor). A 40 karakteres sorméret (szimpla sor) két módon idézhető elő:

- A képernyőre íráskor (pl. PRINT) a kiírt szöveg hosszát 80-al osztva, az osztási maradék 41-nél kisebb. Az írást carrige return zárja le.

- Begépeléssel a képernyőre írva, a kurzor mozgatásával elhagyjuk a sort, mielőtt a gépelés elérné a 40 karaktert.

Ha visszatérünk a kurzorral egy szimpla sorra és a gépelés most hosszabb, mint 39 jel, a sor mérete megváltozik, dupla sor lesz. A sorméret megváltozásával együtt a képernyő alsó sora eltűnik. Az INST, DEL billentyű a sorμérettel összhangban működik.

Igényesebb képernyő kezeléshez ismernünk kell a sorokról tárolt információkat a zéro lapon:

$d1/$d2:	aktuális sor kezdetének mutatója
$d3 :	a kurzor oszlop pozíciója
$d5 :	aktuális sor tényleges hossza
$d6 :	Kurzor sor pozíciója
$d8 :	inzertek száma az aktuális sorban
$d9-$f2:	a sorkezdetek mutatója, együtt a folytatás jelzésével (MSB).

Térjünk vissza az írás előtti kurzor pozicionálásra:

POKE214,S:POKE211,0:SYS58640

Ha a kijelölt sor dupla sor, a sor mérete 80 karakter. Így pozicionálhatunk pl. a kijelölt sor 60. oszlopára: POKE214,3:POKE211,60. A kurzor nem a kijelölt sorban, hanem a következő sor 20. oszlopán jelenik meg.

Ha a kijelölt sol szimpla sor vagy folytatás sor és így pozicionálunk a 60. oszlopra, az MSB nincs összhangban a sormérettel. A pozicionálás után kiírt szöveg egy része meg sem jelenik, ami megjelenik, az is máshol, mint szerettük volna.

Egy soron belül maradva elegendő csak az oszlop pozíció változtatása a kurzor pozicionálásához: POKE211,0.

Idézőjel mód

A vezérlő karakter kiadása a PRINT, INPUT, PRINT# utasításokban lehetséges CHR$(v) (v: vezérlő karakter ASCII kódja) formában. A vezérlő karakterek beépítésének van egy rövidebb módja is. A vezérlő karaktereknek is van "képe" és karakterként beépíthetők stringbe. Ez azonban csak idézőjel módban

lehetséges. Idézőjel mód: az utasításban szerepel az " jel. Az első " jel beállítja az idézőjel módot. A következő " jel zárja le az idézőjel módot. A két jel között begépelt kurzor vezérlő, ill. szín vezérlő karaktereknek a képe kerül a stringbe, a vezérlés nem hajtódik végre. A stringet kiíró utasítás (pl. PRINT) fogja a vezérlést elvégezni.

BILLENTYŰZET

A billentyűzetet a megszakító rutin olvassa. Ha lenyomott billentyűt érzékel, a billentyűnek megfelelő kódot a RAM-ban erre a célra szolgáló FIFO pufferben helyezi el további felhasználásra. A "felhasználó" lehet a BASIC interpreter, esetleg egy általunk írt assembler rutin.

A billentyűzet olvasása és a beolvasott kód felhasználása befolyásolható, többnyire BASIC-ben is.

A billentyűk egy 8*8 vonalú mátrix keresztezési pontjain vannak. A lekérdezést a CIA 1 A és B pontja végzi az alábbi ábra szerint:

port B, bit

port A, bit	0	1	2	3	4	5	6	7
p 0	del	return		F1	F3	F2	F5	↑
o 1	3	W	A	4	Z	S	E	shift bal
r 2	5	R	D	6	C	F	T	X
t 3	7	Y	G	8	B	H	U	V
A 4	9	I	J	0	M	K	O	N
' 5	+	P	L	-	.	:	@	,
b 6	£	*	;	home	shift jobb	=		/
i 7	1	←	ctrl	2	space	C=	Q	run stop
t								

A SHIFT LOCK billentyű a baloldali shift billentyűvel van összekötve, igy külön vonalat nem igényel.

A RESTORE billentyűt sem tartalmazza a mátrix. A billentyű megnyomása nem maszkolható (NMI) megszakítást vált ki.

A billentyűk megnyomásának megfelelő kódot 4 táblázat tartalmazza a ROM-ban. A billentyű helye a 8*8-as mátrixban jelöli ki a megfelelő táblázatban az ASCII értéket. A kódtáblázatokban szereplő $ff érték érvénytelen kombinációt jelöl. (Pl. a CTRL és a + billentyű együttes megnyomása hatástalan.) A táblázatok:

a/ Shift nélkül megnyomott billentyű, címe: $EB81

$14	$0D	$1D	$88	$85	$86	$87	$11
$33	$57	$41	$34	$5A	$53	$45	$01
$35	$52	$44	$36	$43	$46	$54	$58
$37	$59	$47	$38	$42	$48	$55	$56
$39	$49	$4A	$30	$4D	$4B	$4F	$4E
$2B	$50	$4C	$2D	$2E	$3A	$40	$2C
$5C	$2A	$3B	$13	$01	$3D	$5E	$2F
$31	$5F	$04	$32	$20	$02	$51	$03

b/ Shifttel együtt megnyomott billentyű, címe: $EBC2

$94	$8D	$9D	$8C	$89	$8A	$8B	$91
$23	$D7	$C1	$24	$DA	$D3	$C5	$01
$25	$D2	$C4	$26	$C3	$C6	$D4	$D8
$27	$D9	$C7	$28	$C2	$C8	$D5	$D6
$29	$C9	$CA	$30	$CD	$CB	$CF	$CE
$DB	$D0	$CC	$DD	$3E	$5B	$BA	$3C
$A9	$C0	$5D	$93	$01	$3D	$DE	$3F
$21	$5F	$04	$22	$A0	$02	$D1	$83

c/ Commodore billentyűvel együtt megnyomott billentyű, címe: $EC03

$94	$8D	$9D	$8C	$89	$8A	$8D	$91
$96	$B3	$B0	$97	$AD	$AE	$B1	$01
$98	$B2	$AC	$99	$BC	$BB	$A3	$BD
$9A	$B7	$A5	$9B	$BF	$B4	$B8	$BE
$29	$A2	$B5	$30	$A7	$A1	$B9	$AA
$A6	$AF	$B6	$DC	$3E	$5B	$A4	$3C
$A8	$DF	$5D	$93	$01	$3D	$DE	$3F
$81	$5F	$04	$95	$A9	$92	$AB	$83

d/ CTRL billentyűvel együtt megnyomott billentyű, címe: $EC78

$FF	$FF	$FF	$FF	$FF	$FF	$FF	$FF
$1C	$17	$01	$9F	$1A	$13	$05	$FF
$9C	$12	$04	$1E	$03	$06	$14	$18
$1F	$19	$07	$9E	$02	$18	$15	$16
$12	$09	$0A	$92	$0D	$0B	$0F	$0E
$FF	$10	$0C	$FF	$FF	$1B	$00	$FF
$1C	$FF	$1D	$FF	$FF	$1F	$1E	$FF
$90	$06	$FF	$05	$FF	$FF	$11	$FF

Tekintsük át még egyszer a billentyűzet kezelés software folyamatát. Közben kitérünk a "beavatkozás" lehetőségére. Mint említettük, a billentyűzetet a megszakító rutin olvassa és egy pufferbe helyezi a kódot. A puffer 10 karakteres, helye: 631-640. A pufferben lévő feldolgozatlan karakterek száma a 198-as címen található. A 650-es című rendszerváltozó a karakter ismétlést befolyásolja:

- ha értéke 64: egyik billentyű sem ismétel
- ha értéke 128: minden billentyű ismétel
- ha értéke 0: a szóköz és a kurzor vezérlő billentyűk ismételnek (alaphelyzet).

A beolvasást, ill. az utána következő kódkonverziót BASIC-ben nem tudjuk befolyásolni. Ha saját kódtáblázatokat kívánunk használni, erre van lehetőség, de e miatt a megszakító rutint át kell írni (címe a $0314/$0315 byte-ban van). Erre azért van szükség, mert a kódtáblázatok címét másként nem tudjuk megváltoztatni (a rutin minden megszakításkor újra tölti).

A FIFO puffert BASIC-ben is olvashatjuk:

```
WAIT198,1
X=PEEK(631)
POKE198,0
```

A pufferből kiolvasott karakternél már végezhetünk konverziót vagy ellenőrzést BASIC-ben is. Pl.:

```
5   S$=""
10  WAIT 198,1
20  X=PEEK(631)
30  POKE198,0
40  IF X=13 THEN GOTO 100
50  IF X<48 OR X>57 THEN GOTO 10
60  S$=S$+CHR$(X)
70  GOTO 10
100 REM A SZÁM FELDOLGOZÁSA
```

A fenti példa egy számot olvas be. A beolvasás közben a kurzor nem gyullad fel. A beolvasott karakter visszaírásáról is kell gondoskodnunk a feldolgozásnál.

Ha az ellenőrzés a fentinél bonyolultabb, a beolvasás BASIC-ben lassú lesz. A puffer olvasásra adunk assembler példát is. A beolvasott byte az akkumulátorban áll rendelkezésre, lehetőséget ad a további feldolgozásra.

```
10 read lda $c6
20 sta    $cc
30 beq    read
40 sei
50 jsr    $e5b4
          . további
          . feldolgozás
          .
```

Fenti rutinnál olvasás közben a kurzor villog. Ha a kurzor zavar, a 20 és 30 számú sort cseréljük fel, így a kurzor nem gyullad fel.

A billentyűzet mechanikusan össze van építve a géppel. Az elektromos kapcsolat egy 20 tűs csatlakozón keresztül történik. A csatlakozó felhasználásával másik billentyűzet, esetleg párhuzamosan kapcsolt billentyűzet is könnyen csatlakoztatható. Adatrögzítő programoknál szokásos párhuzamosan kapcsolt numerikus billentyűzet használata. Természetesen billentyűzet helyett pl. több tucat kapcsoló megnyomása is lekérdezhető.

PRINTEREK

A számítógép professzionális felhasználása nyomtató nélkül elképzelhetetlen, de a komolyabb amatőrök se igen nélkülözhetik azt. Mindazonáltal a nyomtatás - akár a legjobb nyomtatóval is - az egyik leglassúbb és legtöbb hibalehetőséget tartalmazó számítógépes művelet, maguk a nyomtatók pedig viszonylag drága perifériák. Emiatt a megfelelő printer kiválasztása egy konfiguráció összeállításakor nem egyszerű feladat.

Personal vagy home computerekhez ma kétféle printert használnak,a mátrixnyomtatót és a margarétakerekes nyomtatót.

Néhány korábban alkalmazott rendszer, pl. a termálpapíros nyomtató,kiszorult a gyakorlatból.

A margarétakerekes nyomtató tulajdonképpen egy számítógépvezérlésű írógép. Levélminőségű írásképe - és magas ára is - ennek felel meg. Nevét a betűkerék alakjáról kapta. Működése lassú, kifejezetten levelek vagy sokszorosításra szánt anyagok írására szolgál. A C 64-hez illeszthető ugyan margarétakerekes nyomtató, de erre nem térünk ki, mert ára többszöröse a C 64 árának.

I Ezeket elsősorban szövegfeldolgozó (word processing) rendszerekben I
I alkalmazzák, erre a feladatra azonban a C 64 - a képernyő viszonylag I
I alacsony felbontása miatt - nem nagyon alkalmas. I

A mátrixnyomtató működési elve, hogy egy 7-9 tűből álló, függőleges tűsort vízszintesen mozgat a papír előtt; a tűk egy-egy pontot üthetnek a papírra, s így tetszőleges pontmátrix jeleníthető meg.

A printerek gyakran nincsenek egy adott rendszerhez kötve, a gyártók törekvése az, hogy termékeiket minél többféle típusú számítógéppel lehessen használni. Ezt többek között az elterjedt interface szabványok (CENTRONICS, RS232) alkalmazásával érik el.

I A C 64-hez a C 64 PLUS bővítőkártya tartalmaz CENTRONICS interface I
I software-t (a USER PORT-on keresztül), csak egy csatlakozó kell hozzá. I
I Az RS232 adapter szintén beszerezhető. I

A C 64 "házi" rendszere a soros IEC busz. Erre közvetlenül, minden kiegészítés nélkül csatlakoztathatók a COMMODORE nyomtatók, valamint a VC típusjelű SEIKOSHA mátrixnyomtatók. (Pl. SEIKOSHA GP100 VC.)

Az EPSON típusú printerek opcionálisan tartalmazzák az IEEE-488 interface-t.

A printerekkel kapcsolatban fontos kérdés a működési sebesség, írásminőség, megbízhatóság, sorszélesség, valamint az egyéb szolgáltatások. Ezek értékeléséhez közlünk egy összehasonlító táblázatot, 8 printer jellemző adataival.

Nyomtató típus:	SEIKOSHA GP-100 VC	GP-250 X	GP-700 A
nyomtatási sebesség (kar/sec)	30	50	50
papírméret (inch)	4.5-10	10	10
(karakter)	80	80	108
levélpapír befűzhető?	nem	nem	nem
max. pld.	2	3	3
papírvég észlelés?	nincs	van	van
lapozás?	nincs	van	van
karakterkészlet	2 (nagy- és kisbetűk)	4	4
felhasználói karakter	van	van	van
karakter méret, stílus	dupla széles, inverz	dupla széles, dupla magas	kétfele nyújtott
szinezés	nincs	nincs	7 szín
közel levélminőség?	nem	nem	nem
függőleges tabulálás	nincs	van	van
illesztés - soros	IEC	RS232-C	RS232-C
- párhuzamos	-	CENTRONICS	CENTRONICS

Nyomtató típus:	CBM 1526,MPS 802	MPS 803	EPSON RX 80
nyomtatási sebesség (kar/sec)	80	60	80
papírméret (inch)	4.5-10	10	10
(karakter)	80	80	80
levélpapír befűzhető?	igen	igen	igen
max. pld.	3	3	3
papírvég észlelés?	van	van	van
lapozás?	van	nincs	van
karakterkészlet	2 (nagy- és kisbetűk)		11 féle
felhasználói karakter	van	van	van
karakter méret, stílus	dupla széles, inverz		zsugorított, nyújtott,dőlt, kiemelt 2 betűtípus
szinezés	nincs	nincs	nincs
közel levélminőség?	igen	nem	nem
függőleges tabulálás	nincs	nincs	van
illesztés - soros	IEC	IEC	RS232-C
- párhuzamos	-	-	CENTRONICS IEE-488

Nyomtató típus:	EPSON FX80	EPSON FX100
nyomtatási sebesség (kar/sec)	160	160
papírméret (inch) (karakter)	10 80	16 132
levélpapír befűzhető?	igen	igen
max. pld.	3	3
papírvég észlelés?	van	van
lapozás?	van	van
karakterkészlet	8 féle	8 féle
felhasználói karakter	a teljes karakterkészlet szabadon átdefiniálható	
karakter méret, stílus	zsugorított, dőlt, proporcionális dupla széles, kiemelt, aláhúzott karakter, kétféle betűtípus	
szinezés	nincs	nincs
közel levélminőség?	igen	igen
függőleges tabulálás	van	van
illesztés - soros - párhuzamos	RS232 CENTRONICS IEEE-488 IEC	RS232 CENTRONICS IEEE-488 IEC

C 64 HARDWARE

Bevezetés

A C 64 felépítését nem tárgyaljuk részletesen. A hangsúlyt a gép és környezetének kapcsolatára helyezzük.

A C 64 fő építőelemei

MPU 6510

8 bites mikroprocesszor. Az adatbusz 8 bites, a cím busz 16 bites. Az órajel kb. 985 kHz. A processzorba 6 bites I/O port van beépítve. A port a RAM 0-as és 1-es byte-ján keresztül érhető el. A 0-as byte az adatírány regiszter, az 1-es byte az adat regiszter.

A processzor veremként a RAM $100-1FF területét használja.

SID 6581

Három külön-külön programozható oszcillátor van beépítve. Hangonként négy keverhető rezgés, három keverhető szűrő, burkoló generátor segíti a hangképzést. Tartalmaz még két gyűrűs modulátort és két 8 bites A/D átalakítót.

A chiphez külső jelforrás is csatlakoztatható és a jel keverhető.

VIC 6569

VIC: video kontroller
16 szín használható. A képernyőt 40*25 karakterre bontja. Használható 320*200 pontos grafika is. 8 független sprite definiálható. A karakter generátor és a video RAM a tárban eltolható.

CIA 6526 2 db

CIA: Complex Interface Adapter
A chipnek 16 külön programozható I/O vonala van. 2 független timer és egy 24 órás programozható óra van beépítve. Beépített 8 bites léptető regiszter a soros I/O-hoz.

AM 825100

AM: Address Manager
16 bemenettel és 8 kimenettel rendelkezik. A RAM, a ROM-ok cím kiválasztását végzi.

Tárak

- BASIC ROM : 8 Kbyte
- KERNAL ROM : 8 Kbyte
- Karakter ROM : 4 Kbyte
- Szín RAM : 1 Kbyte
- Általános RAM : 64 Kbyte

A C 64 címbusz 16 bites, 64 Kbyte érhető el. A RAM és a ROM-ok között tehát cím átfedés van. A megfelelő átkapcsolást az Address Manager végzi software úton. A szokásos tárelrendezés, melyet BASIC-ben programozva használunk:

1. RAM

- $0000 - $03FF: rendszerváltozók, verem, stb. (mindig be van kapcsolva)
- $0400 - $07FF: képernyő RAM (áthelyezhető más RAM területre is)
- $0800 - $9FFF: Basic terület
- $A000 - $BFFF: BASIC nem használja, "felette" van a BASIC interpreter ROM
- $C000 - $CFFF: szabadon használható RAM, a BASIC terület ide is átcímezhető a megfelelő rendszerváltozókkal (a 4 Kbyte programozáshoz kicsi, a standard BASIC területtel pedig nem lehet összekapcsolni, így BASIC-ben nem használatos)
- $D000 - $DFFF: az I/O rendszer használja, ill. a chipek regiszterei vannak itt, BASIC-ben nem használható
- $E000 - $FFFF: BASIC nem használja, "felette" van a KERNAL ROM

2. ROM

- $A000 - $BFFF: BASIC interpreter
- $D000 - $DFFF: karakter ROM
- $E000 - $FFFF: KERNAL

3. Szín RAM

$D800 - $DBFF: címzése teljesen külön történik

A fenti elrendezésből látszik, hogy a RAM-ból 21 Kbyte BASIC-ben semmire sincs használva. Ha a modul porton ROM bővítést használunk (pl.HELP+, C64+, stb.), újabb 8 Kbyte RAM (a modul alatti 8 Kbyte) válik használhatatlanná. Persze csak BASIC-ben. A file struktúráknál látjuk, hogy ezeken a területeken memória file-okat létesíthetünk.

Kivezetések

A C 64 hardware vonatkozásban "nyílt" gép. A chipek lábainak jelentős része kapcsolódik a gép valamelyik kivezetéséhez. Ezeken a kivezetéseken keresztül a gép épitőelemeit igen sok célra használhatjuk. (Pl. összekapcsolhatunk több gépet, terminálként használhatjuk nagyobb gépekhez, felhasználhatjuk a házi riasztóberendezés vezérlésére, stb.).

Témánk szempontjából a legfontosabb a C 64-hez nem rendszeresített perifériák működtetése. Ezek az illesztések általában a USER portra, vagy a modul (más néven bővítő) portra csatlakoznak. A kivezetések programozása lényegében a CIA chipek regisztereinek programozását jelenti. A CIA-k címe és regiszterei:

CIA 1	CIA 2	név	feladata
$DC00	$DD00	PORTA	I/O kapu A
$DC01	$DD01	PORTB	I/O kapu B
$DC02	$DD02	DDRA	adatírány regiszter A
$DC03	$DD03	DDRB	adatírány regiszter B
$DC04	$DD04	T1L	Timer 1 alsó byte
$DC05	$DD05	T1H	Timer 1 felső byte
$DC06	$DD06	T2L	Timer 2 alsó byte
$DC07	$DD07	T2H	Timer 2 felső byte
$DC08	$DD08	DOD10	napi óra, 1/10 sec
$DC09	$DD09	TODS	napi óra, sec
$DC0A	$DD0A	TODM	napi óra, perc
$DC0B	$DD0B	TODM	napi óra, AM/PM
$DC0C	$DD0C	SDR	léptető regiszter soros I/O-hoz
$DC0D	$DD0D	ICR	megszakítás kontroll regiszter
$DC0E	$DD0E	CRA	kontroll regiszter A
$DC0F	$DD0F	CRB	kontroll regiszter B

A két CIA I/O kapui közül a CIA 2 B kapuja és az A kapu 2. bitje szabad, a többi foglalt. A szabad kapu a USER portra van kivezetve. A USER port láb kiosztása:

föld	A - 1	föld
CIA FLAG vonal	B - 2	+5V
CIA port B 0	C - 3	RESET
CIA port 1	D - 4	CIA 1 CRA 5. bit soros kimenet számláló
CIA port 2	E - 5	CIA 1 CRA 6. bit soros I/O
CIA port 3	F - 6	CIA 2 CRA 5. bit soros kimenet számláló
CIA port 4	H - 7	CIA 2 CRA 6. bit soros I/O
CIA port 5	J - 8	CIA 2 PC vonala
CIA port 6	K - 9	soros busz ATN vonala
CIA port 7	L - 10	9V AC max. 100 mA
CIA PORTA 2.	M - 11	9V AC, az előzővel ellenkező fázisban
föld	N - 12	föld

A C 64 "belsejéhez való hozzáférés másik lehetősége a modul port. A láb kiosztás:

föld	A + 1	föld
ROMH	B + 2	+5V
RESET	C + 3	+5V
NMI	D + 4	IRQ
02	E + 5	R/W
A15	F + 6	DOT CLOCK
A14	H + 7	I/01
A13	J + 8	GAME
A12	K + 9	EXROM
A11	L + 10	I/02
A10	M + 11	ROML
A9	N + 12	BA
A8	P + 13	DMA
A7	R + 14	D7
A6	S + 15	D6
A5	T + 16	D5
A4	U + 17	D4
A3	V + 18	D3
A2	W + 19	D2
A1	X + 20	D1
A0	Y + 21	D0
föld	Z + 22	föld

A portra tehát ki vannak vezetve a címvonalak (A0-A15), az adatvonalak (D0-D7) és a vezérlővonalak. A porton a kommunikáció külső CIA chipek csatlakoztatásával is megoldható. A RAM-ban "meg van a helye" a külső CIA regisztereinek ($DE00-$DEFF és $DF00-$DFFF).

Ha a port I/01 vonala 0, ez a $DE00, az I/02 vonal a $DF00 címet jelöli ki.

A port és egy külső CIA tipikus kapcsolása:

```
6526        modul port

            VCC <---> +5V
<- PA0      RS0 <---> A0
<- PA1      RS1 <---> A1
<- PA2      RS2 <---> A2
<- PA3      RS3 <---> A3
<- PA4      RES <---> RES
<- PA5      DB0 <---> D0
<- PA6      DB1 <---> D1
<- PA7      DB2 <---> D2
<- PB0      DB3 <---> D3
<- PB1      DB4 <---> D4
<- PB2      DB5 <---> D5
<- PB3      DB6 <---> D6
<- PB4      DB7 <---> D7
<- PB5       02 <---> 02
<- PB6     FLAG <------------| 3.3k |-----------o+5V
<- PB7       CS <---> I/O1
<- PC       R/W <---> R/W
<- TOD      IRQ <---> IRQ
            VSS <---> föld
```

A modul porton történő illesztés szokásos megoldása a külső CIA használata (pl. párhuzamos busz kezelése).

A USER portra illesztett perifériánál a CIA2 B kapuja használatos. Alábbi példánkban itt illesztünk egy CENTRONICS interfésszel rendelkező nyomtatót. A C 64 és a nyomtató összekapcsolásához szükséges kábellel a következő kapcsolatot kell létrehoznunk a USER port és a nyomtató CENTRONICS bemenete között:

USER port		CENTRONICS
A	föld	16
B	FLAG-BUSY	11
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	PA2-STROBE	1

Az átvitelhez szükséges programot assemblerben adjuk meg:

```
        *=$c000
        lda #addrl<
        ldy #addrl>
        sta $0320       ; CHKOUT
        sty $0321       ; vektor
        lda #addr2<
        ldy #addr2>
        sta $0326       ; OUTPUT
        sty $0327       ; vektor
        lda #$ff
        sta $dd03
        lda $dd02
        ora #4
        sta $dd02
        rts
addrl   jsr $f30f       ;  logikai file-szám keresése
        beq printl      ;  megvan
        jmp $f701       ; "FILE NOT OPEN"
printl  jsr $f31f       ;  file-paraméterek
        lda $ba
        cmp #5             a készülékszám 5?
        beq print2         igen
        jmp $f25b       ;  vissza a CHKOUT-hoz
print2  jmp $f275       ;  output kijelölés
addr2   pha
        lda $9a         ;  a készülékszám = 5?
        cmp #5
        beq print3      ; igen
        jmp $flcd       ; vissza a CHROUT-hoz
print3  lda $b9
        and #15
        bne print4
        stx $02
        pla
        tax
        lda $c100,x     ;  a nyomtatandó karakter
        ldx $02
        .byte $24
print4  pla
        pha
        sta $dd01
        lda #$10
print5  bit $dd0d       ;  kiadás a USER porton
        beq print5
        lda $dd00
        ora #4
        sta $dd00
        and #$fb
        sta $dd00
        rts
```

A rutint a sys49152 utasítással aktivizálhatjuk. A nyomtatás előtt OPENlf,5 utasítással kell a nyomtató file-t megnyitni. A karakter konverzióhoz szükséges értékeket a 49408 címtől kell elhelyeznünk a használt nyomtatónak megfelelően. Ha konverzióra nincs szükség, az alábbi BASIC ciklussal állíthatjuk be a megfelelő értékeket:

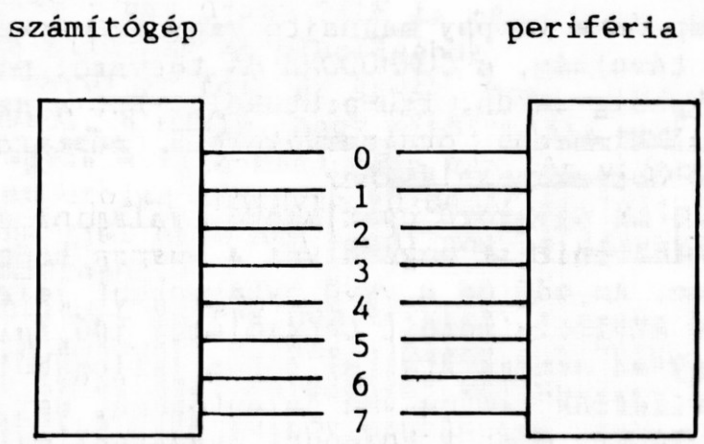
```
        FOR I=49408 TO 49408 + 255
        POKE I,I-49408
        NEXT
```

A COMMODORE DRIVE-OK

SOROS ÉS PÁRHUZAMOS BUSZ

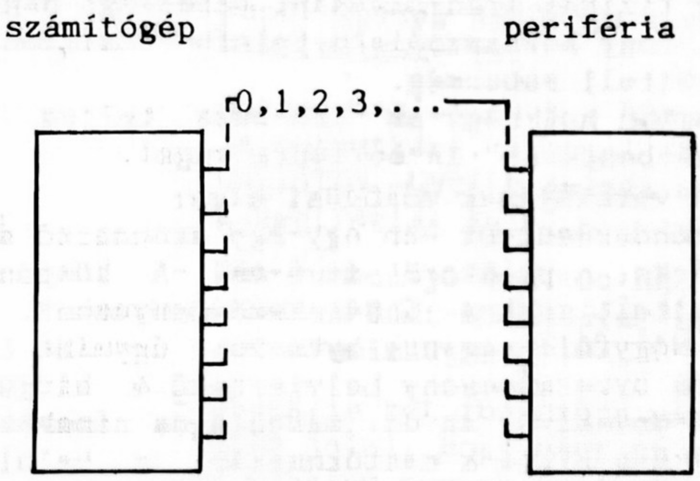
A (byte szervezésű) számítógép és a külvilág közötti kommunikáció leggyakrabban az alábbi két séma valamelyike szerint zajlik:

1. Párhuzmos átvitel



A nyolc átviteli vonalon az adatbitek egyidőben haladnak át.

2. Soros átvitel



Az adatbyte bitjei egymás után haladnak át egyetlen átviteli vonalon.

A mai mikroprocesszoros rendszerek belső adatátvitele mindig párhuzamos, ezért az adó oldalán konverzió szükséges a bitsorozat előállításához, a vevő oldalán pedig ugyanennek a fordítottjához. Az átviteli idő is (legalább) 8-szoros a párhuzamos átvitelhez képest. Soros adatátvitelhez viszont csak egy átviteli vonalra van szükség.

A gyakorlatban általában soros átvitelt alkalmaznak 100 lábnál (kb. 30 méter) nagyobb távolságra történő adatátvitel esetén, valamint számítógép-modem kapcsolatokban.

Utóbbira széles körben elterjedt szabvány az RS-232. (RS - Recommended Standard, ajánlott szabvány, 232 a bejegyzési száma, közölte az EIA 1969-ben. Későbbi módosításait betűvel jelzik, pl. RS-232 C.)

A szabvány egy terminál (vagy mikroszámítógép) [DTE - Data Terminal Equipment] és egy ún. kommunikációs eszköz [DCE - Data Communication Equipment] közötti adatforgalmat írja le. Elterjedtségével függ össze, hogy olyan perifériákon is alkalmazzák, amelyek nem tekinthetők kommunikációs eszköznek, pl. mátrixnyomtatók.

A szabvány rögzíti a berendezéseket összekötő kábel csatlakozóit (sőt, ezek lábkiosztását is [ISO dokumentum]). Ez a DT25 25 lábú csatlakozó. Ilyet a C 64-en nem találunk. Emiatt - habár az RS-232 teljesen software támogatását tartalmazza - az alapgép NEM RS-232 kompatibilis.

Egy otthoni számítógép és a floppy meghajtó vagy nyomtató között aligha van 100 lábnál nagyobb távolság, a COMMODORE 64 tervezői mégis a soros átvitelt választották, mégpedig az ún. IEC-protokolt. Ezt a szabványt többnyire a méréstechnikában alkalmazzák programvezérelt műszerek egymással, ill. számítógéppel történő összekapcsolásához.

Előnye az olcsó kábel, az egyszerű csatlakozó, valamint az a sajátossága, hogy nagymértékű aszinkronitást engedélyez a buszra kapcsolt berendezések átviteli sebességében. Az adó és a vevő byte-onként jelzi, hogy kész a kommunikációra, két byte átvitele között tetszőleges idő telhet el. (Hozzá kell azonban tenni, hogy ez nem az átvitel soros jellegéből következik.)

Ennek intelligens perifériák esetén van jelentősége, egy ilyen perifériának ui. más "dolga" is lehet, mint a központi egységgel való kommunikáció.

Az IEC-busz komoly hátránya, hogy rendkívül lassú, az átviteli sebesség 0.4 kbyte/sec.

Ez alig jobb, mint a ZX Spectrum és a kazettás magnó közötti átviteli sebesség. Összehasonlításképpen megemlítjük, hogy az 5 1/4 inch, szimpla sűrűségű floppy fizikai írás-olvasási sebessége nagyobb, mint 21 kbyte/sec, valamint, hogy közhasználatú telefonvonalakon is elérhető 1.2 kbyte/sec adatátviteli sebesség.

A lassúság oka elsősorban az, hogy az IEC-busz teljes időzítését, sőt az adatbyte "bitekre bontását" is software végzi.

Az IEC-busz COMMODORE-verziójának működési elve:

A buszra kapcsolt berendezéseknek van egy-egy azonosító száma, pl. az (egyik) floppy egysége a 8-as, a printeré a 4-es. A központi egységnek nincs egységszáma. Adatátvitelt csak a C 64 kezdeményezhet. Ezt parancs byte-ok kiadásával teszi. Négyféle parancs byte van, úgymint LISTEN, TALK, UNLISTEN, UNTALK. A parancs byte alacsony helyiértékű 4 bitje az egységszám. Vezérlő szerepük van ezenkívül az ún. másodlagos címeknek. A másodlagos cím alacsony helyiértékű 4 bitje a csatornaszám, a baloldali félbyte lehet OPEN vagy CLOSE kód vagy más.

A központi egység és a periféria OPEN/CLOSE eljárása különböző, csak nem független folyamat. A központi egység a logikai file-számmal azonosítja a file-t, a periféria a másodlagos címmel.

Az adatvitel folyamatát egy példával, egy programfile betöltésével szemléltetjük: (Az egyszerűség kedvéért feltételezzük, hogy a file már meg van nyitva.)

A központi egység kiadja a TALK kódot (pl.) a 8-as egységszámmal, mintegy "megszólítja" a 8-as egységet: adatokat vár tőle (TALK=beszél). Ezt a kódot természetesen még a többi egység is veszi. A C 64 ezután egy ún. másodlagos címet küld a buszon. Az "intelligens" floppy meghajtó e másodlagos címmel azonosítja a file-t: mivel egy meghajtón több file lehet nyitva, a

központi egységnek közölnie kell, hogy melyikből kíván olvasni. A másodlagos cím kiküldése után a soros busz többi egysége (pl. a 4-es és a 9-es) várakozási állapotba kerül az adatátvitel végéig. A 8-as egység "előveszi" a file következő byte-ját. Lehet, hogy ehhez be kell olvasnia egy újabb szektort. Eközben a központi egység, miután jelezte, hogy ő, a hallgató, kész az adat fogadására, várakozik. A meghajtó, ha megvan a keresett byte, lekérdezi, hogy a hallgató kész-e az adat fogadására, s ha igen, jelzi,hogy ő viszont kész az adat küldésére és elkezdi bitenként átvinni a byte-ot.

Ha ez megtörtént, megint veszi a következő byte-ot és vár a "hallgató kész az adat fogadására" jelzésre.A hallgató - ezúttal a központi egység - olvashat egy újabb byte-ot vagy UNTALK jelzés kiadásával befejezheti a kommunikációt. Később - egy újabb TALK és másodlagos cím kiadásával visszatérhet ugyanennek a file-nak az olvasásához.

Két kérdés merülhet fel, az egyik, hogy mi történik, ha a megszólított egység nem válaszol, a másik, hogy miből tudja meg a hallgató, hogy egy közleménynek (vagyis a file-nak) vége van. Az utóbbi esetben az történik, hogy a beszélő az utolsó adatbyte küldését késlelteti, azaz a "beszélő kész az adat küldésére" jel után nem kezdi meg az adatküldést, hanem vár egy kicsit (cca. 250 microsec.).

Ha a megszólított egység nem létezik: a soros busz áramköri megoldásából következően a központi egység először azt "hiszi", hogy nemcsak létezik, de már kész is az adat küldésére; később azt, hogy a file utolsó byteját akarja küldeni; s csak amikor ezután sem érkezik adat, akkor "látja be", hogy DEVICE NOT PRESENT, a keresett egység nincs a soros buszon.

Az IEC-busz 6 vezetéket tartalmaz, ezek közül egyet a perifériák nem használnak. Valamennyi 0-aktív és - a RESET kivételével - ún. "open collector" kapcsolású.

RESET	- A C 64 belső RESET-vonalára van kapcsolva.A központi egység RESET-je kiváltja a perifériák inicializálását is.
ATTENTION	- Ezt a vonalat csak a központi egység használhatja outputként. A vonal aktív, ha LISTEN, TALK, UNLISTEN, UNTALK jelzés vagy másodlagos cím megy a perifériák felé.
CLOCK	- Két funkciója van: az adó (beszélő) aktiválja,ha kész az adat küldésére, később a soros átvitelt szinkronizálja.
DATA	- Ennek is két funkciója van: a vevő (hallgató) ezen jelzi, hogy kész az adat fogadására és ezen a vonalon mennek az átvitel bitjei.
SERVICE REQUEST IN	- A C 64-ben a "rend kedvéért" be van kötve (egy felhúzó ellenállással a CIA#1 chip FLAG lábára), a perifériák azonban nem használják ezt a vonalat.
GROUND	- rendszer föld

Bármelyik egység a soros buszon, ha nincs várakozási állapotban,de nem használja a buszt (ez akkor adódik, amikor valamilyen parancsot hajt végre, pl. töröl egy file-t), a CLOCK vonal 0-ra állításával megakadályoz minden átvitelt a buszon.

Ezt semmi sem indokolja. Éppen egy parancs végrehajtása közben semmi szüksége nincs a buszra, fel kellene szabadítania, lehetővé téve a többi egységgel való kommunikációt.

A COMMODORE a 64-esnél nagyobb gépein nem használja az IEC-buszt.IEEE-488 jelzéssel kifejlesztett egy párhuzamos átviteli vonalat, amely funkcionálisan hasonló az IEC-hez, de annál háromszor gyorsabb.

IEEE - Institut for Electric and Electronic Engeneering, Villamos és Elektronikus Fejlesztések Intézete, egy szabványügyi szervezet (USA).

A C 64 az IEEE-488 interfacet sem hardware, sem software oldalról nem támogatja. Több olyan bővítő modul is forgalomban van, amely tartalmazza a párhuzamos busz kezelését.

INTELLIGENS PERIFÉRIA

Adatokat küldeni külső tárolóeszközre, vagy arról adatokat olvasni bonyolult, többszintű feladat, sőt feladatrendszer.

Bizonyos feladatok - a legfelső szinteken - függetlenek a külső eszköz működési elvétől, típusától, stb., az alsóbb szintek feladatai azonban egyre inkább berendezésfüggőek.

A külső tárolóeszközt (perifériát) attól függően mondjuk többé vagy kevésbé intelligensnek, hogy e berendezés-orientált feladatok mekkora részét vállalja át a központi egységtől.

E feladatok - jelentősen egyszerűsítve - öt körbe sorolhatók:

- a központi egység és a periféria közötti átviteli vonal vezérlése,
- a perifériát vezérlő információk kiértékelése. Ide tartozik pl. a nyomtató vezérlő karaktereinek leválasztása a nyomtatandó szövegről vagy lemeznél a sáv-szektor kijelölés ellenőrzése,
- az adatállományok logikai vezérlése. Pl. a nyomtató puffer vezérlése vagy lemezes állománynál szabad hely keresés, bejegyzés a directoryba, adatszektorok láncolása, stb.
- írás-olvasás logikai vezérlése. Pl. a nyomtató a karakter ASCII-kódját pontmátrixszá alakítja, a lemezegység ellenőrző byte-okat képez,
- a periféria fizikai vezérlése. Pl. a floppy író-olvasó fej léptetése, a papírtovábbító mechanizmus beindítása stb.

A drive maga is egy mikroszámítógép (6502-es processzorral, 16k ROM, 2-4k RAM tárolóval. A dual meghajtókban a Disc Controller funkcióját külön processzor, egy 6504-es veszi át, amely a 6502-es koprocesszoraként működik.

Software architektúrájuk három részre (task-ra) tagolódik, amelyek time sharing-ben dolgoznak. Az ütemezést a megszakítás rutin végzi. Megszakítást vált ki a központi egység ATTENTION szignálja, valamint 14 ms-onként az időzítő. Ha az ATN okozott megszakítást, a 8 byte hosszú buszkezelő rutinra kerül a vezérlés, amely beállít egy flaget és befejeződik. A másik két task ennél összetettebb. Az időzítő alulcsordulása a Disc Controllert indítja el. A többi funkció egy várakozási ciklusra van felfűzve, ez a várakozási ciklus a főprogram. Sorrendben a következőket végzi el:

- megvizsgálja, érkezett-e kérés a központi egységtől a busz kiszolgálására (ATN). Ha igen, a buszkezelő rutinra adja a vezérlést.
- Ha a buszon parancs érkezett, a parancs kiértékelő-végrehajtó programrészt aktiválja.
- Számba veszi az ún. aktív job-okat, két drive-os meghajtónál drive-onként.

  (Aktív job: nyitott csatorna (file), vagy be nem fejezett Disc Controller tevékenység.)

I Ha nincs aktív job, kikapcsolja a LED-et, és némi várakozás után a I
I meghajtó motorját. I
I A Disc Controller feladata szektorok írása-olvasása a lemezre, vala- I
I mint a lemez inicializálása. I

DRIVE TÍPUSOK

	1541	2031	4040
Meghajtók száma	1	1	2
Író-olvasó fejek száma	1	1	1
Formattált kapacitás (Kbyte)	175	175	2x175
Szabad szektor	664	664	2x664
Sávkapacitás (szektor)	17-21	17-21	17-21
KAPACITÁS:			
Szekvenciális file (max, Kbyte)	168	168	168
Relatív file (max.)	167	167	167
Sávok száma	35	35	35
Adathordozó (5 1/4 inch)	SS,SD	SS,SD	SS,SD
Interface	soros	IEEE-488	IEEE-488
ELÉRÉSI IDŐK: (milliszekundum)			
Sávon belül max.	200	200	200
Fejpoz. szomszéd sáv	30	30	30
Fejpoz. max.	360	360	360
Átviteli sebesség (Kb/s)	0,4	1,2	1,2
Belső átviteli sebesség (Kb/s)	40	40	40
Meghajtó software	DOS 2.6	DOS 1.0	DOS 2.1
Pufferek száma	5	5	13
RAM (kbyte)	2	2	4

	8050	SFD1001	8250
Meghajtók száma	2	1	2
Író-olvasó fejek száma	1	2	2
KAPACITÁS:			
Formattált kapacitás (Kbyte)	2x533	1066	2x1066
Szabad szektor	2x2102	4133	2x4133
Sávkapacitás (szektor)	23-29	23-29	23-29
Szekvenciális file (max, Kbyte)	521	1050	1050
Relatív file (max.)	183	1040	1040
Sávok száma	77	2x77	2x77
Adathordozó (5 1/4 inch)	SS,DD	DS,DD	DS,DD
Interface	IEEE-488	IEEE-488	IEEE-488

	8050	SFD1001	8250
ELÉRÉSI IDŐ: (milliszekundum)			
Sávon belül max.	200	200	200
Fejpoz. szomszéd sáv	5-30 (*)	6	6
Fejpoz. max.	125-750 (*)	125	125
Átviteli sebesség (Kb/s)	1,2	1,2	1,2
Belső átviteli sebesség (Kb/s)	40	40	40
Meghajtó software	DOS 2.5	DOS 2.7	DOS 2.7
Pufferek száma	13	13	13
RAM (Kbyte)	4	4	4

*: A 8050-es egység tartalmazhat Tandon vagy Micropolis meghajtót. A két meghajtó fejpozícionálási sebessége jelentősen eltér. (A Tandon a gyorsabb.)

SS - Single Sided, egyoldalas floppy
DS - Double Sided, kétoldalas
SD - Single Density, szimpla írássűrűségű (40 track/inch)
DD - Double Density, dupla sűrű (min. 77 track/inch)

Winchesterek:

	D9060	D9090
Meghajtók száma	1	1
Író-olvasó fejek száma	4	6
KAPACITÁS:		
Formattált kapacitás (Mbyte)	4.98	7.47
Szabad szektor	19442	29162
Cilinderkapacitás (szektor)	128	192
Szekvenciális file (max, Mbyte)	4.94	7.41
Relatív file (max.)	4.90	7.35
Cilinderek száma	153	153
Interface	IEEE-488	IEEE-488
ELÉRÉSI IDŐK: (milliszekundum)		
Cil-en belül max.	16.7	16.7
Fejpoz. szomszéd cil.	3	3
Fejpoz. max.	153	153
Átviteli sebesség (Kb/s)	1,2	1,2
Belső átviteli sebesség (Mb/s)	5	5
Meghajtó software	DOS 3.0	DOS 3.0
RAM (Kbyte)	4	4

KOMPATIBILITÁS

A floppy meghajtókkal kapcsolatban két különböző értelemben beszélünk kompatibilitásról.

1. Azt mondjuk, hogy két - különböző típusú - egység kompatibilis, ha az egyik egységen felírt lemezt a másik egység is tudja olvasni (és írni) és fordítva.
2. Más értelemben mondunk kompatibilisnek két egységet akkor, ha az egyik egység jelenlétét feltételező programok változtatás nélkül működnek a másik egységgel és fordítva.

Ez a kétféle kompatibilitás egymástól csaknem teljesen független, egyik sem zárja ki a másikat, de nem is következnek egymásból.

Ad 1.

A COMMODORE floppy meghajtók három kompatibilitási osztályba sorolhatók:

- 1541,2031,4040

E szimpla írássűrűségű floppyk formátuma azonos, a három egység teljesen kompatibilis.

- 8050

Egyoldalas, dupla sűrű. Korlátozottan kompatibilis a kétoldalas dupla sűrű meghajtókkal, ld. a következő bekezdést.

- SFD1001,8250

Kétoldalas, dupla sűrű "egymegások". A 8250 két meghajtós egység, ettől eltekintve azonosak. A két meghajtó képes olvasni az egyoldalas 8050-en felírt lemezeket is. (Az első hozzáfordulás egy 8050-en felírt lemezhez 66,ILLEGAL TRACK OR SECTOR hibaüzenetet okoz, amit figyelmen kívül lehet hagyni, a többi művelet hibátlanul fut le.)

A gyári dokumentációk fordított irányú kompatibilitásról is beszélnek. Ezt, tekintve, hogy csak akkor áll fenn, ha a 8250-en vagy az SFD-n felírt lemeznek csak az egyik (az alsó) oldalán van adat, a gyakorlatban nem kell komolyan venni.

Ad 2.

Programozási szempontból a COMMODORE drive-ok (ideértve a Winchestereket is) **csaknem teljesen** kompatibilisek. Kompatibilitási kérdésekben egy ilyen "csaknem" alatt többnyire sok vesződséget kell érteni, de mint látni fogjuk, itt az eltérések valóban minimálisak:

- a különböző kapacitású egységeken - értelemszerűen - eltér a BAM (Block Allocation Map, szektor foglaltsági térkép) helye és némileg a felépítése is.
- Eltér a tartalomjegyzék (directory) kapacitása:
  1541, 2031, 4040: max. 144 file
  8050, 8250, SFD1001: max. 232 file
  D9060, D9090 (Winchester): korlátlan
  A tartalomjegyzék felépítése azonban mindegyiknél azonos.
- Értelemszerűen eltér a sávok száma és a sávon belül a szektorok száma különböző kapacitású egységeken.
- A 8050-en egy relatív file maximális mérete 182 kbyte. A többi egységen a relatív file betöltheti a teljes lemezt.
- Ha a szekvenciális (vagy program) file-t nyitunk outputra (vagy SAVE-vel mentjük) úgy, hogy azonos nevű file már létezik, akkor az SFD1001, 8250, D9060, D9090 egységeken

```
OPEN 3,8,4,"0:@ADAT,S,W"
```

a többi egységen

```
OPEN 3,8,4,"@0:ADAT,S,W"
```

a szintaxis. (SAVE-nél hasonlóképpen.)

> Itt jegyezzük meg, hogy a 1541-es egységen az íly módon megnyitott (vagy SAVE-elt) file-ok időnként zavart okoznak: felülírnak egyéb létező file-okat a lemezen. A hibát eddig nem sikerült azonosítanunk, de valószínű, hogy a drive software hibája.

- Szintén értelemszerű, hogy a BACKUP (teljes lemez másolás) parancs csak a két meghajtós egységeken (4040,8250) működik.

Az utóbbihoz kapcsolódik, hogy egy meghajtós egységeken több esetben el lehet hagyni a meghajtó kijelölést. Pl. 1541-es készüléken.

```
OPEN 2,8,15:PRINT#2,"N0:JATEKLEMEZ,JL"
```

helyett működik az

```
OPEN 2,8,15:PRINT#2,"N:JATEKLEMEZ,JL"
```

is. Ne éljünk ezzel a lehetőséggel!

> Két előnyt szerzünk, ha nem élünk vele. Egyrészt a programot nem kell módosítani, ha pl. 1541-es helyett 4040-es egységgel futtatjuk, másrészt elkerüljük az alábbihoz hasonló csapdákat (áldozatának hosszú fejtörést okozott):
>
> ```
> 10 OPEN 2,8,15
> 20 OPEN 3,8,4,"RELFIL,L,"+CHR$(58)
> 30 INPUT#2,EC,T$
> 40 IF EC>=20 THEN PRINT"HIBA:";EC;T$:CLOSE 2:END
> ```
>
> Meg akar nyitni egy relatív file-t, 58 byte-os rekordhosszal, látszólag kifogástalanul. Az eredmény mégis
>
> ```
> HIBA: 34 SYNTAX ERROR
> ```
>
> A CHR$(58) ui. véletlenül éppen a kettőspont kódja, s a drive az első kettőspont után keresi a file nevét, előtte a drive számot. Meg kell adni a drive számot:
>
> ```
> 20 OPEN 3,8,4,"0:RELFIL,L,"+CHR$(58)
> ```
>
> s a hiba megszünik.

PARANCSOK

A drive parancsok alkotják az intelligens COMMODORE lemezegységek utasításkészletét. Fontos, hogy ezeket világosan megkülönböztessük a BASIC input-output utasításoktól. Az utóbbiak logikai file-okat kezelnek, tehát többé-kevésbé készülékfüggetlenek, képesek értelmezni és végrehajtani azokat.

A parancsokat a 15-ös másodlagos címre (csatornára) megnyitott logikai file-on küldjük a központi egységtől a drive-nak. Ugyanezen a file-on olvashatjuk vissza a parancs "sorsáról" szóló információkat. (Sikeres volt-e a végrehajtás, ha nem, milyen hiba akadályozta.) E kettős szerepe miatt nevezik a 15-ös csatornát parancscsatornának és hibacsatornának is. A központi egység a parancs küldése után várakozás nélkül folytatja munkáját, hibacsatorna olvasás esetén azonban természetes módon vár a parancs végrehajtás befejeződésére.

A parancsokat számos kiadvány, dokumentáció közli, de szinte valamenynyi hiányos és/vagy hibás. Sajnos, a leggyengébbek közé éppen a (meghajtókkal együtt szállított) gyári dokumentációkat kell sorolnunk.

A parancsok szintaktikája nem egységes. A paramétereket általában karakteresen ábrázolt számként kell megadni, néhány parancsnál pedig bináris formában (chr$(n) alakban).

A karakteres ábrázolású számként megadott paraméterek között az elválasztójel szóköz vagy vessző (",").

A parancsokat két csoportban ismertetjük:

1. NEW
   INITIALIZE
   VALIDATE
   COPY
   RENAME
   SCRATCH
   BACKUP

1.1 NEW

Az új lemezt használatba vétel előtt **NEW** paranccsal meg kell formázni. A NEW hatására a meghajtó felírja az összes sávot, a lemez nevét,ID-jét, a BAM-ot és egy üres tartalomjegyzéket.

A meghajtó a lemeznevet nem, csak az ID-et használja egy lemez azonosítására.

Szintaxis:

NEWd: név [,id]
Nd: név [,id]

d - a drive száma, "0" v. "1"
nev - a lemez neve (max. 16 hosszú)
id - a lemez ID-je

Ha az ID-t elhagyjuk, csak a BAM-ot és a tartalomjegyzéket írja fel a lemezre. (Korábban már használt lemez törlését végezhetjük így el.)

Példák:

```
1/ OPEN 1,8,15,"N0:SOFTWARE,SW"
2/ OPEN 1,8,15
   PRINT#1,"N0:SOFTWARE,SW"
3/ OPEN 15,8,15,"N1:UJSOFTWARE"
```

Megjegyzés: Ha a 1541-es egységen valamely okból a NEW hibásan fut le, akkor újabb kisérlet előtt az egységet kapcsoljuk ki és újra vissza. Ha programból szeretnénk újraindítani a megszakadt NEW-t, a következő parancsot kell kiadni előtte:

```
PRINT#1f,"M-W"CHR$(81)CHR$(0)CHR$(1)CHR$(255)
```

1.2 INITIALIZE

A lemez BAM-ját olvassa be az egység memóriájába. Az egység ezt automatikusan is megteszi bizonyos parancsok végrehajtása előtt, ha először fordul a lemezhez.

Mégis indokolt ezt mindig külön kiadni, mert
- nem minden parancs előtt automatikus (pl. BLOCK-ALLOCATE előtt nem),
- azonos ID-vel rendelkező lemezeknél "nem veszi észre" a rendszer a lemezcserét, s az előző lemez BAM-jával dolgozik tovább, ha az INITIALIZE elmarad. (A hatás értékelését az Olvasóra bízzuk.)
- az "elhagyott" (lekötött, de fel nem szabadított) puffereket felszabadítja,
- végül, lehetnek az egységnek olyan hibaállapotai, amit csak az INITIALIZE szüntet meg (a ki-bekapcsolás sem).

I Pl., ha a 1541-es egységen egy nem létező sávra, mondjuk a 36-osra pozícionált a fej. Innen "nem talál vissza" a directory sávba, csak az INITIALIZE hibajavító eljárásának segítségével. I

Szintaxis:

INITIALIZd
Id

ahol d - drive szám "0" v. "1"

1.3 VALIDATE

A megnyitott, de le nem zárt file-ok területét felszabadítja és törli a rájuk vonatkozó bejegyzést a tartalomjegyzékből. Ugyancsak felszabadítja azokat a blokkokat (szektorokat), amelyek egy file-hoz sincsenek hozzárendelve. Az így újra létrehozott BAM-ot visszaírja a lemezre.

Szintaxis:

VALIDATEd
Vd

ahol d - drive szám "0" v. "1"

Megjegyzés:
- direkt file-okat tartalmazó lemezre nem szabad VALIDATE-et kiadni
- ha a VALIDATE bármely okból megszakad vagy nem fejeződik be, a lemezt ne használjuk addig, amíg a hibát ki nem küszöböltük és a VALIDATE jól le nem fut.

1.4 COPY

A COPY segítségével file-okat másolhatunk egy lemezegységen belül. Lehetőség van arra is, hogy több input file-t összekapcsolva hozzunk létre egy output file-t.

Szintaxis:

```
COPYdo:ofile=[di:]ifi 1[,ifi 2]...
Cdo:ofile=[di:]ifi 1[,ifi 2]...
```

ahol do - az output drive "0", v. "1"
ofile - output file név
di - input drive. Elhagyható, ha megegyezik az output drive-val.
ifil1,ifil2,... - input file(ok). Ha több van, akkor ezek egymás után kapcsolódva kerülnek az output file-ba.

Használható relatív file másolásra is, ekkor csak egy input lehet. Programfile-okat nem lehet összekapcsolni. Az input file-ok számát csak az korlátozza, hogy egy parancs-string nem lehet 40 byte-nál hosszabb.

1.5 RENAME

Egy file nevét változtathatjuk meg a RENAME segítségével. A file néven kívül a file tulajdonságai és tartalma változatlan marad.

Szintaxis:

```
RENAMEd:újnév=réginév
```

ahol d - drive szám, "0" v. "1"

1.6 SCRATCH

File(ok) törlését váltja ki. A file számára lefoglalt blokkok felszabadulnak.

Szintaxis:

```
Sd:file1[,file2]...
```

ahol d - drive szám, "0" v. "1"
file1,file2,... - törölni kívánt file név(ek)

Megjegyzés:
- A file névben használhatjuk az ún. jokereket. Pl.:

```
OPEN 7,8,15
PRINT#7,"S0:TEMP*,?/*"
```

Törli a "TEMP"-pel kezdődő file-okat (pl. TEMP7 vagy TEMPORARY), valamint azokat, amelyeknek második karaktere "/". (Pl. C/ROGZIT vagy Z/EDI)

- A hibacsatornán a

  01,FILES SCRATCHED,nn,00

  olvasható ki, ahol nn a törölt file-ok száma. (Lehet 0 is.)

2. BLOCK-READ
   BLOCK-WRITE
   BUFFER-POINTER
   BLOCK-ALLOCATE
   BLOCK-FREE

   P

   Memory-Write
   Memory-Read
   Memory-Execute
   BLOCK-EXECUTE
   USER

2.1 BLOCK-READ

Adatblokkot olvas a lemezről egy adott file-hoz rendelt pufferbe. A pufferből vehetjük át az adatokat a C 64 memóriájába, a következő konvenciók szerint:

A puffer első byte-ja azt mutatja, hogy hány értékes byte van a pufferben. Ezt nem tudjuk beolvasni. Ezt követően az első byte által meghatározott számú byte olvasása után file vége (EOI, End-Or-Identify) állapot következik be. (ST=64)

Szintaxis:

```
BLOCK-READ:cs,d,sáv,szektor
B-R:cs,d,sáv,szektor
```

ahol cs - a (direkt) file-hoz rendelt csatorna (másodlagos cím)
d - drive szám, "0" v. "1"

Példa:

```
10 OPEN 15,8,15:OPEN 2,8,7,"#"
20 TR=17:S=0
30 PRINT#15,"B-R:7,0"TR;S
```

A 17-es sáv 0-as szektort olvassa.
A 30-as sor néhány lehetséges más alakja (hatásuk a példában szereplő sorral azonos):

```
30 PRINT#15,"B-R:7 0 17 0"
30 PRINT#15,"B-R:"7;0;TR;S
30 SA=7:D=0:PRINT#15,"B-R:"SA;D;TR;S
30 PRINT#15,""BLOCK-READ:7,0"TR;S
```

2.2 BLOCK-WRITE

Adott file pufferét írja fel a lemezre. A puffer pointert a blokk első byte-jába írja.

Szintaxis:

```
BLOCK-WRITE:mc,d,sáv,szektor
B-W:mc,d,sáv,szektor
```

Ld. 2.1. BLOCK-READ. A puffer pointerrel kapcsolatban ld. a direkt file-okról szóló részt a File típusok c. fejezetben.

2.3 BUFFER-POINTER

Egy adott puffer pointerét állítja be. Ld. File típusok: direkt file.

Szintaxis:

```
BUFFER-POINTER:mc,pérték
B-P:mc,pérték
```

ahol mc - az a csatorna, amelynek puffer pointerét állítjuk.
pérték - a puffer pointer új értéke.

Megjegyzés: A gyakorlatban direkt file-ok kezelésénél használjuk, elvben azonban tetszőleges file típusnál alkalmazható.

Példa:

```
PRINT#lf,"B-P:"2;0
```

A 2-es csatorna pufferének pointerét a puffer elejére állítja.

2.4 BLOCK-ALLOCATE

A megadott blokkot a BAM-ba lefoglalja és a BAM-ot visszaírja a lemezre.

Szintaxis:

```
BLOCK-ALLOCATE:d,sáv,szektor
B-A:d,sáv,szektor
```

ahol d - a drive szám "0" v. "1"
sav,szektor - a kért blokk

Amennyiben a kért blokk már foglalt, a hibacsatornán

```
65,NO BLOCK,tt,ss
```

üzenet olvasható, ahol tt,ss a kért blokk utáni első szabad blokk. Ha nincs szabad blokk a kért blokk után, tt=ss=0.

2.5 BLOCK-FREE

Hatása a BLOCK-ALLOCATE-énak a fordítottja, egy adott blokkot a BAM-ban felszabadít és a BAM-ot visszaírja.

Szintaxis:

BLOCK-FREE:d,sáv,szektor
B-F:d,sáv,szektor

Ld. BLOCK-ALLOCATE.

Megjegyzés: A blokk felszabadítása megtörténik tekintet nélkül arra, hogy azt ki, mikor, miért foglalta le, tehát alkalmazása elővigyázatosságot igényel. Ha a jelzett blokk már korábban is szabad volt, az utasításnak nincs hatása.

2.6 P

Relatív file adott sorszámú rekordjára pozícionál.

Szintaxis:

P mc ah mh pp

mc - az a másodlagos cím, amelyre a relatív file-t megnyitottuk
ah,mh - rekordsorszám alacsony, ill. magas helyiértékű byte-ja
pp - pozíció a rekordon belül, l<= pp <= rekordhossz

Példa:

PRINT#19,"P"CHR$(12)CHR$(149)CHR$(1)

Ld. ehhez File típusok, a relatív file c. részt.

2.7 Memory-Write

A drive RAM-jába írhatunk a Memory-Write segítségével. Erre akkor van szükség, ha felhasználói programot akarunk a drive RAM-jában elhelyezni, vagy a drive változóit felül szeretnénk írni, pl. a készülékszám megváltoztatása érdekében.

Szintaxis:

M-W ah mh nb b1 b2...bn

ah,mh - annak a címnek alacsony ill. magas helyiértékű byte-ja, ahova írni akarunk
nb - a beírható byte-ok száma, max. 34 (ekkora a puffer)
b1,..,bn - a beírandó byte-ok

Példa:

```
OPEN 7,8,15

PRINT#7,"M/W"CHR$(0)CHR$(2)CHR$(2)CHR$(24)CHR$(96)
```

A drive memóriájában a $200-as a címen a

```
CLC
RTS
```

utasításokból álló bonyolult programot helyezi el.

2.8 Memory-Read

Egy byte-ot olvashatunk Memory-Read utasítás segítségével a lemezegység memóriájából (RAM-ból vagy ROM-ból).

Szintaxis:

M-R ah mh

ah,mh - az olvasandó byte címének alacsony ill. magas helyiértékű byte-ja.

A Memory-Read-del megcímzett byte a parancscsatornán olvasható be.

Példa:

```
OPEN 7,8,15
PRINT#7,"M-R"CHR$(215)CHR$(254)
GET#7,A$
PRINT "A LEMEZ SAVKAPACITASA:";ASC(A$)-1
```

A lemezegység ROM-jából, a $FED7 címről olvasott egy byte-ot.

2.9 Memory-Execute

A lemezegység memóriájában levő tetszőleges szubrutin hívható Memory-Execute-tal. A szubrutinnak RTS-el kell befejeződnie.

Szintaxis:

M-E ah mh

ah,mh - a szubrutin címének alacsony, ill. magas byte-ja

Példa:

```
OPEN 7,8,15
PRINT#15,"M-E"CHR$(0)CHR$(3)
```

A lemezegység memóriájában a JSR $0300 című szubrutin.

2.10 BLOCK-EXECUTE

A lemezről egy gépi kódú rutint tölt be egy pufferbe, majd szubrutin hívással ugrik a puffer elejére.

Szintaxis:

```
BLOCK-EXECUTE:mc,d,sav,szektor
B-E:mc,d,sav,szektor
```

mc - csatornaszám
d - a meghajtó száma (0 vagy 1)
sav,
szektor - a gépi kódú rutint tartalmazó blokk címe

A rutinnak RTS-el kell befejeződnie.

2.11 USER

Tíz USER utasítás van,(részben) eltérő funkciókkal.

U1,
UA - szektor olvasása
U2,
UB - szektor írása
U3-U8,
UC-UH - ugrás felhasználói rutinra
UI - ugrás NMI rutinra
UJ - RESET

Megjegyzések:
- Az U1, U2 parancsok használata megegyezik a B-R, ill. B-W parancsokkal, hatásuk azonban nem pontosan! A B-R és B-W helyett használjuk az U1 és U2 parancsot. (A DOS 2.1-ben a B-R, B-W hibás.)
- Az U3-U8 parancsok ugrási címei:

Lemezegység RAM mérete	2K	4K
U3	$500	$1300
U4	$503	$1303
U5	$506	$1306
U6	$509	$1309
U7	$50C	$130C
U8	$50F	$130F

- Az NMI funkció készülékenként eltér. NMI-t általában a lemezegység hardware generál, az NMI rutin helyén normál szubrutin található.
- 1541-es lemezegységre küldött UJ (reset) parancs után, annak befejeződéséig (1 sec) a soros buszon semmilyen művelet nem végezhető.

Várakozás az UJ parancs befejeződésére 1541-es készülék esetén:

```
a/ PRINT#7,"UJ"
   FOR I=1 TO 1000: NEXT
b/ PRINT#7,"UJ":WAIT 56576,128
```

FILE TÍPUSOK

Ebben a fejezetben a C 64 drive-okon elérhető standard file típusok BASIC programokból való kezelését írjuk le.

Szekvenciális file-ok

A programfile-ok (PRG) és a user (USR) típusú file-ok is szekvenciálisak.

Szekvenciális file-ok megnyitása:

```
OPENlf,un,mc,"{0}:filenév,{S},{W}"
              {1}          {P} {R}
                           {U} {A}
```

lf: logikai file-szám
un: készülékszám
mc: másodlagos cím (0-14)
S: SEQ
P: PRG
U: USR
R: read (alapértelmezés)
W: write
A: append

A drive hibacsatornáját kiolvasva meggyőződhetünk, hogy sikerült-e a megnyitás.
Pl.:

```
10 OPEN 4,8,15
20 OPEN 1,8,2
30 INPUT#4,HK,H$
40 IF HK >= 20 THEN PRINT HK;H$:STOP
```

Lehetséges hibák:

63, FILE EXISTS

A lemezen van már ilyen nevű file.

72, DISK FULL

A tartalomjegyzék tele van, az új bejegyzésnek nincs már hely, vagy a lemezen egy szabad szektor sincs.

A file első szektorát a DOS a megnyitáskor foglalja le, s a szektor címét a file névvel együtt beírja a tartalomjegyzékbe.

70, NO CHANNEL

A drive nem talál szabad puffert a file számára, az egységen "túl sok" file van nyitva.

2k RAM-ot tartalmazó lemezegységeken (ld. DRIVE-ok táblázat) egyszerre 3 szekvenciális file lehet nyitva, 4k-snál 5. Egy relatív file 2 szekvenciális file-t "ér", két direkt file egyet.

Természetesen felléphetnek más hibák is. A hibacsatorna kiolvasását soha ne hagyjuk el.

Megjegyzés:
- Az 1-es csatorna programfile írására használt
- a 0-as csatorna programfile olvasására használt

Programfile megnyitása:

```
Pl:  OPEN 7,8,1,"0:PROG,P,W"
```

PROG egy programfile (a "P" jelzi). Az előző utasítás egyszerűbben:

```
OPEN 7,8,1,"0:PROG"
```

Elhagyható tehát a ",P,W". Az

```
OPEN 7,8,1,"0:ADATOK,S,W"
```

viszont hibás. (64 FILE TYPE MISMATCH hibaüzenetet vált ki) Programfile-ok keletkeznek a SAVE utasítás hatására is.

A user (USR) file-okról az irodalomban nem található semmi információ, mégsem titokzatosak, ugyanis mindenben megegyeznek a SEQ típusú file-okkal.

```
Pl:  OPEN 7,8,9,"0:FELH,U,W"
```

utasítással nyithatunk user file-t.

A file név max. 16 hosszú lehet, nem tartalmazhatja az alábbi tíz különleges karakter egyikét sem:

```
? @ * = : , $ # CHR$(160) CHR$(13)
```

A megnyitott szekvenciális file-ba PRINT# utasítással írhatunk: Pl:

```
100 PRINT#3,"A megnyitott szekvenciális file-ba";
110 PRINT#3," PRINT# utasítással írhatunk:"
```

és így tovább. Az adatok kiírása után a file-t le kell zárni, különben az összes kiírt adat elvész:

```
120 INPUT#4,HK,H$
130 IF HK>=20 THEN PRINT HK;H$:STOP
140 CLOSE 3
```

A "tökéletes" módszer az volna, ha minden írás után (esetünkben 100-as és 110-es sorban) lekérdeznénk az ST hibaváltozót. Ez azonban nehézkes és lassú, ezt pótolja a hibacsatorna kiolvasása a lezárás ELŐTT, az alábbi megfontolás alapján:

Ha a file írása közben a lemezegységen belül lépett fel hiba, (pl. betelt a lemez vagy hibás egy szektor), akkor a további PRINT#-ek a "levegőbe" mennek és a hibaállapot megmarad, a hibakód lekérdezhető.

Meg kell azonban jegyezni, hogy ez a módszer nem fogja kimutatni azokat a hibákat, amelyek nem a drive-on belül keletkeztek, hanem még átvitel közben, a soros buszon. Tipikus esete ennek az, hogy egy gyári hibás VC 1526-os nyomtató illetéktelenül "belebeszél" a C 64 és a drive közötti kommunikációba. Az ilyen hibát csak az ST státusz változó lekérdezésével szűrhetjük ki. (ST=0, ha nem történt hiba.)

Szekvenciális file olvasása:

Megnyitási példák:

```
10 OPEN 3,8,4,"0:ADATOK"          vagy
20 OPEN 3,8,4,"0:ADATOK,R"        vagy
30 OPEN 3,8,4,"0:ADATOK,S"        vagy
40 OPEN 3,8,4,"0:ADATOK,S,R"
```

A 10-es és a 20-as, valamint a 30-as és a 40-es sorban lévő utasítások hatása azonos. Az "R" (read, olvasás) ugyanis alapértelmezés és ezért elhagyható.

A 10-es és a 30-as sor abban különbözik, hogy a 10-es megnyitja az ADATOK nevű file-t, tekintet nélkül annak típusára, a 30-as sor OPEN-je viszont csak akkor lesz sikeres, ha az ADATOK file SEQ típusú.

Az utolsó megnyitást javasoljuk.

Lehetséges hibák:

62, FILE NOT FOUND

Nincs ilyen nevű file.

70 NO CHANNEL

Nincs szabad puffer a file számára.

60 WRITE FILE OPEN

A file felíráskor nem lett lezárva.

Az olvasás INPUT# vagy GET# utasítással történhet. Az ST jelzi a file végét. (ST=64). Célszerűbb azonban, ha felíráskor valamilyen különleges jelet írunk ki a file végére és olvasáskor ezt használjuk a "file vége" azonosítására.

Az input file-t is zárjuk le az olvasás végén, különben nem szabadulnak fel a file számára lekötött pufferek.

A szekvenciális file belső szerkezete

Egy szektor felépítése:

0-1 byte: A file következő szektorának címe (sáv,szektor), ha van következő. Ha nincs, a 0-as byte értéke 0, az 1-es byte az utolsó "élő" byte pointere az utolsó szektorban.

2-255 byte: 254 adatbyte, értékük tetszőleges.

Direkt file-ok

Direkt file-ok esetén a file kezelés egészében a programozó feladata. Ez - a létrehozandó file logikai szerkezetétől függetlenül - az alábbi tevékenységeket tartalmazhatja:

- puffer kijelölés a drive memóriájában
- a puffer pointer beállítása
- adatátvitel a C 64-től a drive pufferbe
- szabad szektor keresése és lekötése
- a puffer felírása a lemezre
- olvasás a lemezről a pufferbe
- olvasás a pufferből a C 64 memóriájába
- lekötött szektor felszabadítása

Puffer kijelölése

A direkt file megnyitásakor jelöljük ki a puffert:
Pl:

```
10 OPEN 20,8,15,"I":REM PARANCSCSATORNA
20 OPEN 3,8,2,"#":REM DIREKT FILE MEGNYITASA
```

A drive keres egy szabad puffert és leköti a direkt file számára.
Ha nem akarjuk rábízni a választást, kérhetünk egy adott puffert:

```
20 OPEN 3,8,2,"#0"
```

A 0-as puffert rendeli a direkt file-hoz (ha a 0-as puffer szabad).
2k RAM-ot tartalmazó készüléken 5 puffer van (0-5), 4k-son 10 (0-9).
A hibaüzenet, ha a puffert nem sikerült lekötni:

```
70, NO CHANNEL
```

A puffer pointer beállítása

A puffer pointer egy változó a drive memóriájában, amely a puffer soron következő byte-jára mutat.

Ha a C 64 kiír egy byte-ot a direkt file-ba, az a pufferben a pointer által meghatározott címre kerül.

Olvasásnál a szerepe ezzel analóg.

Természetesen egy kiírt vagy beolvasott byte után a pointer 1-gyel nő.

Értékét, ha szükséges, a "B-P" (BUFFER-POINTER) paranccsal módosíthatjuk. Ezt a parancscsatornán kell kiadni. Pl.:

PRINT#20,"B-P"2;0

A "2" az a csatorna, amelyre a direkt file-t megnyitottuk, 0 a pointer kívánt értéke (a puffer elejére mutat).

Adatátvitel a C 64-től a drive pufferbe

Írás a pufferbe: PRINT#... utasítással, a szokott módon.

Szabad szektor keresése és lekötése

"B-A" (BLOCK-ALLOCATE) paranccsal

Pl.:

PRINT#20,"B-A"0;17;0

A 0-as drive 17-es sáv 0-as szektor lefoglalását kéri. A szektor lefoglalása a BAM-ban (Block Available Map, foglaltság térkép) a szektorhoz rendelt bit 0-ra állítását jelenti. Ha a kért szektor már foglalt, a hibacsatornán

"65 NO CLOCK tt ss" üzenet jelenik meg

tt: sáv

ss: a kért szektor feletti első szabad szektor ha nincs ilyen, tt=ss=0

Puffer írása a diszkettre

Pl:

PRINT#20,"U2"2;0;tt;ss

Felírja a 2-es csatornán a 0-as meghajtón a tt sávra az ss szektort.

Megjegyzés: A rendelkezésre álló csaknem valamennyi dokumentáció azt írja, hogy az U2 parancs végrehajtása után a puffer pointer automatikusan 0-ra áll. EZ NEM ÍGY VAN, külön be kell állítani.

Olvasás a diszkettről pufferbe

Pl.:

PRINT #20,"U1"2;0;tt;ss

A paraméterek, mint az írásnál.

Megjegyzés: A puffer pointer automatikusan 0-ra áll.

Olvasás a pufferből a C 64 memóriájába

GET#lf, vagy INPUT#lf utasítással.

Lefoglalt szektor felszabadítása

"B-F" (BLOCK-FREE) parancscsal

Pl.:

```
PRINT#20,"B-F"0;17;0
```

A 0-as meghajtón felszabadítja a 17-es sáv 0-as szektort.

A "$" FILE-OK

Mindenki, aki használt már CBM drive-ot, megtanulta, hogyan kell egy lemez tartalomjegyzékét kilistázni:

```
LOAD"$",8
LIST
```

Ezzel azonban még nem merítettük ki a tartalomjegyzék file-ok összes lehetőségét.

Pl.:

```
LOAD"$B*",8
LIST
```

A listán most csak a B betűvel kezdődő file nevek jelennek meg.

A "$" file egy információs file, amelyet a lemezegység állít össze. A C 64 abban a formában veszi át, ahogy a készülék küldi. Ebben a formában természetesen nincs tárolva a lemezen. Felépítése azonos egy BASIC programéval, ezért is lehet kilistázni.

A LOAD utasítás a 0-as csatornát használja. Ha a "$" file-t nyitjuk meg ezen a csatornán, INPUT# vagy GET# utasítással olvashatjuk.

Relatív file

Relatív file létrehozása:

```
OPEN lf,un,mc,"{0}:filenev,L,"+chr$(rh)
               {1}
```

lf - logikai file
un - egység
mc - másodlagos cím, 2-14
rh - rekordhossz

A megnyitással létrehoztuk a relatív file-t, a file név,a rekordhossz, a file típus bekerült a katalógusba. A file újbóli megnyitásakor a ",L,"+chr$(rh) elhagyható. File-unk pillanatnyilag üres.

Az első szektor rekordja(i) azonban már most is léteznek.

Pozicionálás relatív rekordra

A pozicionáló parancsot ("P" parancs) a parancscsatornán (15-ös másodlagos címen) kell kiadni.

A parancs szintaxisa:

```
PRINT#pf,"P"CHR$(mc)CHR$(ah)CHR$(mh)CHR$(pp)
```

pf - a parancs csatornára megnyitott logikai file
mc - az a csatorna, amelyre a relatív file-t nyitottuk meg
ah, mh - a rekordsorszám alacsony ill. magas helyiértékű byte-ja.
ah=RN-256*INT(RN/256)
mh=INT(RN/256), ha RN a rekordsorszám
pp - pozíció a rekordon belül. 1 <= pp <= rekordhossz

Egy dokumentáció által sugallt tévedés, hogy az utolsó paramétert el lehet hagyni, alapértelmezése 1. Természetesen nem lehet elhagyni: ha lehagynánk, a helyére a parancsstringet záró carriage return, CHR$(13) kerülne.

Példa:

```
OPEN 17,8,15,"I"
OPEN 3,8,4,"0:RELFIL,L,"+CHR$(46)
PRINT#17,"P"CHR$(4)CHR$(2)CHR$(2)CHR$(1)
```

A hibacsatornán (a parancscsatorna és a hibacsatorna ugyanaz) az

50, RECORD NOT PRESENT

olvasható ki. Az 514-es rekordra pozicionáltunk (2*256+2) és ez a rekord még nem létezik. Azáltal jön létre, hogy írunk bele:

```
PRINT#3,RE$
```

Egy relatív rekordot egyetlen PRINT# utasítással kell kiírni. Ha a rekordot nem írjuk végig, jobbról CHR$(0)-kal töltődik fel. A jobb oldali CHR$(0)-k nem olvashatók vissza, ezeket a lemezegység "lenyeli". Ha a rekord nem tartalmaz CHR$(0)-t, ez nem okoz gondot, más esetben a rekordot carriage returnnel kell lezárni.

Az 514-es rekorddal együtt létrejön az összes, nála alacsonyabb sorszámú rekord is. (Tartalmuk CHR$(255) a rekord első byte-ján.)

Előfordulhat, hogy a rekorddal együtt néhány nála magasabb sorszámú rekord is létrejön, ha azzal egy szektoron vannak.

A relatív rekordok tömören vannak tárolva, azaz lehetnek olyan rekordok, amelyek átlépik a szektorhatárt.

Relatív rekord olvasás

A relatív file olvasható szekvenciálisan is, ekkor azonban minden rekord olvasása után file vége jelzést kapunk. (ST=64)

Ha nem szekvenciálisan, hanem a sorszám szerint közvetlenül akarjuk elérni a rekordot, ugyanúgy kell pozicionálni, mint írás előtt. A rekord adattartalmát INPUT# v. GET # utasításokkal olvashatjuk be. (Egy rekordot több utasítással is beolvashatunk, csak a kiírást nem lehet "felaprózni".)

Relatív file-ra vonatkozó hibaüzenetek

50. RECORD NOT PRESENT

Nem létező rekordra pozicionáltunk. Az állomány bővítésekor ez természetesen nem jelent valódi hibát.

51, OVERFLOW IN RECORD

A PRINT# utasítással kiírt adatok hosszabbak, mint a rekordméret, vagy a pozicionáló parancs utolsó paramétere (a pozíció a rekordon belül) nagyobb, mint a rekordhossz. A rekordból "kilógó" adatok elvésznek.

52, FILE TOO LARGE

A pozicionált rekord nem létezik és a lemezen már nincs is elég hely a létrehozására.

Megjegyzések a relatív file használatához:

- Ha COMMODORE 1541-es készüléket használunk, a file-t minden olyan írás előtt zárjuk le és nyissuk meg újra, amely előtt előzőleg olvastunk. Ellenkező esetben a rekord (vagy egy része) elveszhet. (A készülék software hibája.)

- Ha a rekord utolsó mezője olyan tipusú, hogy chr$(0) karaktert tartalmazhat, az OPEN utasításban definiáljuk a rekordhosszt eggyel hosszabbra a mezőkből adódó hossznál. Így az I/O zavartalan lesz. A plusz karakter carrige return lesz, melyet a software automatikusan felír, olvasáskor pedig nem kell vele törődnünk, a tényleges rekordhossznak megfelelően olvasunk. Ha ezt elmulasztjuk, a chr$(0) karakterek beolvasáskor lemaradnak a rekord végéről, helyettük chr$(13) karaktereket kapunk. (A legcélszerűbb minden filenál gondolni a záró carriage returnre.Ebből baj nem lehet.)

- Ha ismerjük a file várható méretét, célszerű a file-t előformázni. Ez azt jelenti, hogy a file-t megnyitjuk, az utolsó sorszámra pozicionálunk,kiírunk egy $FF karaktert, majd a file-t lezárjuk. Így a file kezelő rendszer létrehozza az összes rekord eléréséhez szükséges oldalszektorokat és megformázza az összes rekordot. A rekord megformázása azt jelenti, hogy a rekord első byte-jába $FF kerül, a többi byte-ba pedig $00. Ha nem formázzuk elő a file-t, a bővítés időigényes, mert az oldalszektorok karbantartását is rekordonként kell elvégeznie a file-kezelő rendszernek. (Ill. szektoronként, mert a floppy rutinjai szektoronként végzik a formázást.)

Ha a file-t más módon kívánjuk előformázni (pl. a teljes rekordhosszban space jelenti a rekord logikai NEMlétezését), akkor is a legutolsó rekordot írjuk fel legelőször, majd egytől a végéig. Így a formázás ideje töredéke lesz a csak az 1-től a végéig felírásnak.

Zavaró lehet a szektorhatár átlépésekor megjelenő hibajelző villogás a floppyn, amit akkor kapunk, ha olyan rekordra (szektorra) pozicionálunk, mely még nem formázott. A floppy RECORD NOT PRESENT üzenetet ad, ez a file bővítésekor nem jelent hibát! (Ha előformázott a file, akkor igen.)

- Karbantartáskor egy rekord törlésének jelzésére célszerű az előformázott rekordnak megfelelő adattartalmat használni. Így egy rekord logikai létezésének megállapításához nem kell tudnunk, hogy a rekord még nem létezett, vagy már töröltük. Szükség van erre azért is, mert - analógiát vonva az ISAM file-val - a rekord kulcsának létezését nem tudjuk megállapitani, csak a rekord adattartalmából következtethetünk a rekord logikai létezésére.

Megadjuk az oldalszektorok felépítését, ha valaki közvetlenül akarná kezelni.

Az oldalszektort a DOS arra használja, hogy a hivatkozott sorszám alapján meghatározza a rekord helyét a diszketten.

A szektor felépítése:

0.-1. byte:	következő oldalszektor mutatója ha ez az oldalszektor az utolsó a láncban, akkor az 1. byte a blokkban lefoglalt byte-ok számát tartalmazza
2. byte:	az oldalszektor sorszáma
3. byte:	rekordhossz
4.-15.byte:	az oldalszektorok lemezcíme egy relatív file-hoz max. 6 oldalszektor tartozhat, a lemezcím 2 byte-os, a nem létező oldalszektorok címe helyett 0 áll.

A további byte-okban az adatrekordok szektorcíme van két byte-onként (sáv, szektor).

KERNAL

A C 64 gyári software 3 részből áll, ezek a BASIC interpreter, a karakter ROM és a Kernal-nak nevezett...

Mi is ez a Kernal? Az bizonyos, hogy nem operációs rendszer: pl. nem kommunikál a gép kezelőjével, s egyébként sem "tud" annyit, mint az operációs rendszerek (pl. MP/M, MSDOS).

A Kernal voltaképpen egy rutingyűjtemény. Ezeket a rutinokat (pl.) így csoportosíthatjuk:

- I/O rutinok
- megszakító rutinok
- inicializáló rutinok.

A gép készítői fontosság szerint 3 osztályba sorolták a rutinokat,ezek címzésükben különböznek:

- a rutin címzése a Kernal végén elhelyezett ugrótáblán keresztül történik. Az ugrótábla indirekt ugrást tartalmaz a rutin RAM-ban elhelyezett címére
- címzés az ugrótáblán keresztül, direkt ugrással
- bizonyos rutinoknak semmilyen linkage-e sincs.

A KERNAL UGRÓTÁBLÁJA

Cím	Utasítás	Rutin cím	Funkció
FF81	JMP $FF5B		képernyő inicializálás
FF84	JMP $FDA3		CIA regiszterek inicializálása
FF87	JMP $FD50		RAM teszt, 0-3 lap törlése
FF8A	JMP $FD15		I/O vektorok töltése (standard)
FF8D	JMP $FD1A		I/O vektorok töltése (felhasználói)
FF90	JMP $FE18		Státusz (ST, 90) töltése
FF93	JMP $EDB9		Másodlagos cím küldés a soros buszon LISTEN után
FF96	JMP $EDC7		Másodlagos cím TALK után
FF99	JMP $FE25		"RAM vége" pointer írása/olvasása
FF9C	JMP $FE34		"RAM kezdete" pointer írása/olv.
FF9F	JMP $EA87		Billentyűzet lekérdezése
FFA2	JMP $FE21		soros busz timeout-flag állítása
FFA5	JMP $EE13		soros busz: 1 byte olvasás
FFA8	JMP $EDDD		soros busz: 1 byte írás
FFAB	JMP $EDEF		soros busz: UNTALK
FFAE	JMP $EDFE		soros busz: UNLISTEN
FFB1	JMP $ED0C		soros busz: LISTEN
FFB4	JMP $FE07		Státusz (ST)
FFBA	JMP $FE00		File-paraméterek beállítása OPEN-hez
FFBD	JMP $FDF9		File-névparaméterek állítása OPEN-hez
FFC0	JMP ($031A)	$F34A	OPEN
FFC3	JMP ($031C)	$F291	CLOSE
FFC6	JMP ($031E)	$F20E	CHKIN, input csatorna kijelölése
FFC9	JMP ($0320)	$F250	CHKOUT, output csatorna kijelölése

Cím	Utasítás	Rutincím	Funkció
FFCC	JMP ($0322)	$F333	CLRCHN, input/output kijel. törlés (alapértelmezés visszaállítása)
FFCF	JMP ($0324)	$F157	CHRIN, byte olv. az input csatornáról
FFD2	JMP ($0326)	$F1CA	CHROUT, byte írás az output csatornáról
FFD5	JMP $F49E		LOAD (*)
FFD8	JMP $F5DD		SAVE (**)
FFDB	JMP $F6E4		Timer változó (TI) érték állítása
FFDE	JMP $F6DD		Timer változó (TI) érték kiolvasása
FFE1	JMP ($0328)	$F6ED	Stop-flag lekérdezése
FFE4	JMP ($032A)	$F13E	GET, 1 byte olvas
FFE7	JMP ($032C)	$F32F	CLALL, nyitott file-ok törlése
FFEA	JMP $F69B		Timer változó (TI) aktualizálása
FFED	JMP $E505		Képernyő max. sor-oszlop (25/40)
FFF0	JMP $E50A		Kurzor pozíció beállítás/olvasás
FFF3	JMP $E500		CIA 1 reg. címének olv.

*: A LOAD rutin is a RAM-on keresztül címződik:

```
F49E STX $C3
F4A0 STY $C4
F4A2 JMP ($0330) ; $F4A5
```

**: A SAVE hasonlóképpen:

```
F5DD STX $AE
F5DF STY $AF
F5E1 TAX
F5E2 LDA $00.X
F5E4 STA $C1
F5E6 LDA $01,X
F5E8 STA $C2
F5EA JMP ($0332) ; $F5ED
```

I/O HIBÁK

Az alábbi rutinok végrehajtása alatt következhet be hibaállapot:

OPEN, CHKIN, CHRIN, CHKOUT, CHROUT, LOAD/VERIFY, SAVE

E rutinok hiba esetén a carry-t magasra állítják és az akkumulátorban adják vissza a hibakódot. Ha a $9D címen lévő "mod" változó 6. bitje magas, a hibakód a képernyőn is megjelenik

I/O ERROR#hk

alakban, hk a hibakód.

Értéke:	1 - Túl sok file	OPEN
	2 - A file már nyitva van	
	3 - A file nincs megnyitva	CHKIN, CHKOUT
	4 - A file nem létezik	LOAD/VERIFY
	5 - A készülék nem létezik	(bármelyik rutin adhatja)
	6 - Nem input file	OPEN`
	7 - Nem output file	CHKOUT
	8 - File-név hiányzik	LOAD/VERIFY, SAVE
	9 - Érvénytelen készülékszám	

A RUTINOK KIVÁLTÁSA

Csak azok a rutinok válthatók ki, amelyek címe a RAM-ban, a 3-as lapon található. Az, hogy melyek ezek és a 3-as lapon hol található a címük, az ugrótáblából kiolvasható.

Sajnos, a Kernal belső szubrutinhivásainál nem használja az ugrótáblát, így azok a rutinok, amelyek szerepelnek az ugrótáblában, de nem szerepelnek a 3-as lapon, nem válthatók ki. (Pl. a Kernal RAM-ba másolásával és az ugrótábla módositásával.)

A 3-as lapon kiváltható rutinok átírásával ugy definiálhatunk új rutinokat, hogy ezek BASIC-ben "észrevétlenek" maradnak. A BASIC interpreter "nem veszi észre" a cserét. Így működnek pl. modul portra illesztett párhuzamos busz kezelő bővítések (IEEE-488, C64PLUS, stb.).

I/O RUTINOK

Az I/O rutinokat három csoportban tárgyaljuk. Az első csoport a segéd-,ill. előkészítő rutinok, a második a "kiváltható" rutinok, a harmadik a soros busz nem kiváltható I/O rutinjai.

Előkészítő, segédrutinok

Billentyűzet lekérdezése

A megszakitó rutin segédrutinja. A CIA 1 A és B portja segítségével lekérdezi, van-e megnyomott billentyű (lásd perifériák fejezet). Ha van megnyomott billentyű, ellenőrzi az érvényességét. Ha nem érvényes, a megfelelő 8*8-as mátrixban (kódtáblázat) $FF érték áll, nincs további tevékenység. Ha a megnyomott billentyűk érvényes kódot adnak, az ismétlés ellenőrzése következik (a 650 című rendszerváltozó). Ha a változó 7. bitje magas, az összes billentyű ismétel. Ha 0 és a 6. bit is 0, a kurzorvezérlő billentyűk, DEL, INST és space ismétel. Ha a 6. bit magas, a billentyűk nem ismételnek.

File paraméterek beállítása

A regiszterekben lévő - aktív file-ra vonatkozó - értékeket tölti a paramétereket tároló rendszerváltozókba:

A -> $B8 (logikai file-szám)
X -> $BA (készülékszám)
Y -> $B9 (másodlagos cím)

File-név paraméterek beállítása

A már a regiszterekben lévő értékeket tölti a paramétereket tároló rendszerváltozókba OPEN utasítás előkészítéseként:

A - $B7 (file-név hossza)
X - $BB (a file-név címének alsó byte-ja)
Y - $BC (a file-név címének felső byte-ja)

Status olvasása

A rutin ellenőrzi a készülékszámot ($BA), ha a készülék az RS232 ($BA=2), ennek statusát olvassa ($0297), törli. Az eredeti érték az akkumulátorban marad.

Ha a készülék nem az RS232, a status változót ($90) beállítja.

Status töltése

A statust tárolja ($9D), majd beállítja (lásd előző rutin).

Kiváltható rutinok

OPEN

Használt regiszterek: A,X,Y

Az OPEN rutint meg kell előznie a file-paraméterek beállításának a megfelelő rendszerváltozókba ($B7-$BC, lásd fent).

Az OPEN rutin ezekből veszi ki a megnyitáshoz.

Az OPEN BASIC utasítás a $E219 című beolvasó rutint használja a kiértékeléshez. Ez a rutin állítja be a rendszerváltozók értékét.

Az OPEN KERNAL rutin működését a BASIC OPEN utasítás leírásánál (2.fejezet) részletesen tárgyaltuk.

Egy file megnyitása készülékenként külön rutin.

Ha assemblerben dolgozunk, a file-leíró rendszerváltozók értékét szintén be kell állítanunk. Pl.:

```
5    *=$033c
10   lda #64      ;  a logikai
20   sta $b8      ;  file-szám
30   lda #8       ;  a
40   sta $ba      ;  készülékszám
50   lda #0       ;  a
60   sta $b9      ;  csatornaszám
70   lda #1       ;  a file-név
80   sta $b7      ;  hossza
90   sda #36      ;  "$", a file
100  sta $02      ;  neve
110  lda #02
120  ldy #0
130  sta $bb      ;  a file-név
140  sty $bc      ;  címe
150  jsr $f34a    ;  open
160  rts
```

A példa az alábbi BASIC utasításnak felel meg:

OPEN64,8,0,"$"

Vagyis megnyitja a katalógust olvasásra.

Ez a megnyitási módszer a file-leíró táblázatokba is elhelyezi a paramétereket, tehát a C 64 szempontjából is nyitott a file. Ugyanezt a megnyitást elvégezhetjük úgy is, hogy csak a készüléket utasítjuk a megnyitásra. Pl. a soros buszon levő drive megnyitása:

```
5        *=$033c
30 - 140 ig azonos az előzővel
150      lda $b9      ; a másodlagos cím +
160      ora #$60     ; open
170      sta $b9
180      jsr $f3d5    ; megnyitás a soros buszon
190      rts
```

Ha így nyitottuk meg a drive csatornáját, akkor lezárni is csak hasonló rutinnal lehet. A C 64 "nem tud" a megnyitásról, így egy file-számon keresztül nem tudjuk lezárni. (Ill. megnyithatjuk újra a csatornát bármilyen file-számmal, majd ezt a file-t lezárva már a csatorna is lezárul.)

A 3-as lapon keresztül előtét rutint is írhatunk az OPEN elé. Pl.:

Gyakori, hogy a BASIC programban egy file megnyitásánál a file-hoz azonos logikai file-számot és csatornaszámot rendelünk (OPEN 15,ks,15). A megnyitásokat rövidebben is írhatjuk (OPEN 15,ks), ha a hiányzó paramétert az előtéttel pótoljuk:

```
        *=$033c       ; indító rutin
        lda #cim<
        ldy #cim>
        sta $031a     ; a 3-as lap
        sty $031b     ; cím
        rts

cim     ldx $b8       ; a file-szám =
        stx $b9       ; a csatornaszám
        jmp $f34a     ; vissza az open-hez
```

Ennek a példának nem sok haszna van a gyakorlatban, de az OPEN rutin kiválthatóságát szemlélteti.

Az OPEN rutin helyettesítésével, ill. előtét rutin írásával sok egyéb hasznos feladatot oldhatunk meg.

Például a C 64 PLUS bővítésnél az IOSEL utasítás szolgál a soros/párhuzamos busz kijelölésére. Célszerű a bővítést kiegészíteni úgy, hogy egy negyedik file-leíró táblát is kezeljen, ami a logikai file-számhoz rendelt busz kijelölést tartalmazza. Ennek előnye, hogy a készüléket kezelő I/O utasítás előtt nem kell kijelölni a buszt, az a táblázatból kiolvasható. Ha megírjuk az előtétet, a következő szempontokat kell figyelembe vennünk:

- Az OPEN utasítás BASIC interpretálását is meg kell változtatnunk,itt teremtve lehetőséget a busz kijelölésére.

Másik lehetőség, hogy az OPEN előtti IOSEL utasítás által beállított buszkijelölést használjuk az OPEN előtét rutinjában. Így a BASIC interpreterbe nem kell "belenyúlni".

- A többi I/O rutinnal is gondoskodni kell a buszkijelölő táblázat használatáról.

- Gondoskodni kell bővítésünk és a már működő bővítés kapcsolatáról. MINDEN MÁR MŰKÖDŐ BŐVÍTÉSHEZ ILLESZTETT SAJÁT BŐVÍTÉSNÉL A MŰKÖDŐ BŐVÍTÉSBE KELL VISSZATÉRNI! (Ez a módszer mindig jó, ha csak a C 64 software működik, akkor is.)

A C 64 PLUS-nál elterjedtebb párhuzamos busz kezelő bővítés az IEEE-488. Ez a COMMODORE cég saját fejlesztése. Itt a buszkijelölést SYS utasítással kell megadnunk. A buszkijelölés elhagyásához itt is újabb táblázatot célszerű készíteni.

Az újabb file-leíró táblát karbantartó rutin:

```
        *=$033c         ; inditó rutin
addr    lda $031a       ; aktuális open
        ldy $031b       ; rutin
        sta acima       ; címének
        sty acimf       ; tárolása
        lda #ucim<      ; saját
        ldy #ucim>      ; előtét
        sta $031a       ; rutin
        sty $031b       ; címe
        rts

ucim    ldx $98
        lda $fb         ; a buszkijelölés
        sta $03f2,x     ; a buszkijelölő táblázat
        jsr kapcs       ; az átkapcsolás
        jsr addr        ; a címbeállítás
        jmp (acima)

kapcs   cmp #1          ; soros busz?
        beq soros
        cmp #2
        bne valami
parh    jsr $c8a4       ; párhuzamos busz
valami  rts
soros   jsr $fd15       ;  soros busz
        rts

acima   .byte $00
acimf   .byte $00
```

A példában a busz típus tárolására a $FD címet használtuk, ide az OPEN utasítás szintaxisát kiértékelő előtét rutinnak kell betöltenie a busz típust (1:soros, 2:párhuzamos).

A táblázat a $03f2 címen kezdődik.

A többi I/O rutinhoz is kell előtétet illesztenünk. Szerencsére ezek - eltekintve a 3-as lap címtől és saját előtétünk címétől - azonosak. A busztípust a file-számon keresztül állapíthatjuk meg:

```
        ldx $b8
        txa
        ldx $98
ujat    dex
        bmi nincs
        cmp $0259,x
        bne ujat
        lda $03f2,x
        beq soros
        jsr $c8a4
        jmp cim
soros   jsr $fd15
cim     jsr addr
        jmp (acim)
nincs   rts
```

A rutinban hivatkozott ADDR rutin megegyezik az előzővel (a konkrét címek a 3-as lap címekben különböznek).

Ha a hivatkozott file-szám nincs a táblázatban, a hiba lekezlését nem kell elvégeznünk (NINCS cimke), az I/O rutin majd újra megállapitja és hibaüzenetet ad.

CLOSE

Használt regiszterek: A,X,Y

A CLOSE lezár egy nyitott file-t, ha a file szerepel a nyitott file-ok táblázatában. (Működését a CLOSE BASIC utasításnál leírtuk.)

A CLOSE rutin megváltoztatása általában az OPEN rutin átírásával együtt kerül szóba. Pl. az OPEN rutinnal kezelt negyedik file-leíró táblát itt is kezelnünk kell (ha az OPEN-nél használtuk):

```
        jsr $f314       ; file-szám keresése
        txa
        pha
        beq close       ; megvan
        pla
        jmp (acim)      ; nem volt nyitva
close   lda $03f2,x     ; a busz típus
        cmp #1          ; soros?
        beq soros
        cmp #2          ; párhuzamos?
        beq parh
        pla
        jmp (acim)      ; egyik sem
soros   jsr $fd15
        jmp tabla
parh    jsr $c8a8
tabla   lda $98
        sta $02
        pla
        tax
        dec $02
```

```
        cpx $02
        beq ret
        ldy $02
        lda $03f2,y
        sta $03f2,x
ret     jsr addr
        jmp (acim)
```

A rutinhoz a 3-as lap kezelést nem írtuk meg, az megegyezik az OPEN-nel leírttal.

CHKIN

Használt regiszterek: A,X
Input: X

Valamely - már megnyitott file - input csatornakénti kijelölését végzi a file-hoz rendelt készülék típusától függően. (Pl. a képernyő esetén nem csinál semmit, a soros busz esetén felszólítja a készüléket a "beszédre", stb.)

Működése:

Megkeresi a file-számot a nyitott file-ok táblázatában. Ha nincs benne, "FILE NOT OPEN" üzenetet ad és vége. Ha megtalálta, beállítja a file paramétereket és következik a készüléktől függő tevékenység. A rutin akkor is működtethető, ha a hivatkozott file nincs megnyitva. (Csak a készüléket utasítjuk a csatorna megnyitására.) Pl. a soros buszon lévő drive kijelölése input csatornaként:

```
ldx #8          ; készülékszám
jsr $ffb4       ; TALK
lda #2          ; csatorna
ora #$60
jsr $ff96       ; másodlagos cím TALK után
txa
sta $99 ;       ; készülékszám
rts
```

CHKOUT

Használt regiszterek: A,X
Input: X

Valamely - már megnyitott - file output csatornakénti kijelölését végzi a file-hoz rendelt készüléktől függően. (Pl. a soros busz esetén felszólítja a készüléket a "hallgatásra".)

Működése:

Megkeresi a file-számot a nyitott file-ok táblázatában. Ha nincs benne, "FILE NOT OPEN" üzenetet ad és vége. Ha megtalálta beállítja a file paramétereket és következik a készüléktől függő egyéb tevékenység.

A CHKIN-hez hasonlóan adunk példát a soros buszon lévő drive közvetlen kijelölésére output csatornaként:

```
ldx #9         ; készülékszám
jsr $ffb1      ; LISTEN
lda #3         ; csatorna
ora #$60
jsr $ff93      ; másodlagos cím LISTEN után
txa
sta $9a        ; készülékszám
rts
```

CLRCHN

Használt regiszterek: A,X

Aktív I/O csatornát törli, visszaállítja az alapértelmezés szerinti értékeket (output: képernyő, input: billentyűzet).

Működése:

Ha a készülékszám > 3 (soros busz) UNTALK-ot ad ki a soros buszon. Az aktív input készülék alapállapotát ($9A=3) beállítja. Az aktív output készülék alapállapotát beállítja ($99=0).

LOAD

Használt regiszterek: A,X,Y
Input: A - utasításkód (0:load, 1:verify)
X - cím alacsony byte
Y - cím magas byte

A LOAD kernal rutin működését a BASIC LOAD utasítás leírásánál(2. fejezet) részletesen tárgyaltuk.

Példa:
Ha a LOAD utasítást BASIC programban adjuk ki, a betöltés után a program újra indul. Itt egy példát adunk az újraindítás nélküli programtöltésre:

```
10     *=$033c
20     jsr $aefd  ; "," vizsgálata
30     jsr $ad9e  ; file-név beolvasása
40     jsr $b6a3
50     ldx $22    ; file-név
60     ldy $23    ; címe
70     jsr $ffbd  ; file-név pár. beállítása
80     jsr $aefd  ; "," vizsgálata
90     jsr $b79e  ; készülékszám
100    ldy #0
110    jsr $ffba  ; file pár. beállítása
120    jsr $aefd  ; "," vizsgálata
130    jsr $ad8a  ; töltési
```

```
140     jsr $b7f7  ; cím
150     sta $c4
160     sty $c3
170     lda #0
180     jsr $f4a2  ; load
19o     jsr $ffb7  ; status
200     and #$bf
210     bne e      ;  hiba
220     rts
230 e   ldx #$1d   ; "LOAD ERROR"

240     jmp $a437
```

A rutin adott címre tölt be pl. egy gépikódú programot, vagy használhatjuk a memóriában file-ok tárolására is (lásd file-struktúrák fejezet).

<u>Hívása:</u>

SYS828,"név",k,tc

név: a töltendő program neve
k: a készülék száma, ahonnan tölteni kell
tc: a cím, ahová tölteni kell

SAVE

Használt regiszterek: A,X,Y
Input: A - kezdőcím mutató (0. lap)
X - végcím alacsony byte
Y - végcím magas byte

A SAVE KERNAL rutin működését a BASIC SAVE utasításnál (2. fejezet) részletesen tárgyaltuk.

Példaként a LOAD-nál adott példa ellentettjét adjuk meg, egy gépikódú rutin tárolását.

```
10      *=$033c
20      jsr $aefd  ; "," vizsgálata
30      jsr $ad9e  ; file-név
40      jsr $b6a3
50      ldx $22    ; file-név
60      ldy $23    ; címe
70      jsr $ffbd  ; file-név pár. beállítása
80      jsr $aefd  ; "," vizsgálata
90      jsr $ad8a  ; mentési cím
100     jsr $b7f7  ;  honnan
110     sta $c2
120     sty $c1
130     jsr $aefd  ; "," vizsgálata
140     jsr $ad8a  ; mentési cím
150     jsr $b7f7  ;  meddig
160     sta $af
170     sty $ae
18o     jsr $f5ea  ; save
190     rts
```

A rutin adott címtől, adott címig kimenti RAM tartalmát (pl. gépikódú program, képernyő, stb.)

Hívása:

SYS828,"név",k,c1,c2

név: a kívánt file-név
k: a készülék száma
c1: ahonnét menteni kell
c2: ameddig menteni kell

SOROS BUSZ RUTINOK

LISTEN

Használt regiszterek: A
Input: A

Az akkumulátorban megadott készüléknek parancsot ad, hogy legyen "hallgató".

TALK

Használt regiszterek: A
Input: A

Az akkumulátorban megadott készüléknek parancsot ad, hogy legyen "beszélő".

UNLISTEN

Használt regiszterek: A,X

A "hallgató" állapotú készülékeket alapállapotra utasítja.

UNTALK

Használt regiszterek: A,X

A "beszélő" állapotú készüléket alapállapotba hozza.

CLALL

Használt regiszterek: A,X

Lezárja az összes nyitott file-t (a nyitott file-ok számát mutató rendszerváltozóba /$98/ 0-t tölt). Ez a C 64 szempontjából jelenti a file-ok lezárását.

Ezután végrehajt egy CLRCHN-t, ennek leírását lásd ott.

GET

Használt regiszterek: A,Y
Output: A

Működése:

Egy byte-ot olvas az aktuális input készülékről. Ha a készülék a billentyűzet és a billentyűpuffer üres, ($C6=0), akkor nem vár (tehát nem csinál semmit).

Az egyes olvasó rutinok címei készülékenként:

billentyűzet: $E5B4
RS232 : $F086
képernyő : $F16A
soros busz : $EE13
kazetta : $F179

A rutin működtetése előtt a használt készülékre meg kell nyitnunk file-t,ha az nem a billentyűzet.

CHRIN

Használt regiszterek: A,Y
Output: A

Szintén az aktuális input készülékről olvas egy byte-ot. A GET rutintól csak abban különbözik, hogy a billentyűzet, mint input egység esetén is a képernyőn is megjelenő beütést vár.

CHROUT

Használt regiszterek: A,Y
Input: A

Működése:

Egy byte-ot visz át az aktuális output készülékre. Az egyes rutinok címe készülékenként:

soros busz: $EDDD
képernyő : $E716
RS232 : $F20B
kazetta : $F1E5

A SOROS BUSZ RUTINOK MŰKÖDÉSE

A legfontosabb soros busz kezelő rutinok a LISTEN, TALK, IECIN, IECOUT, UNLISTEN, UNTALK, valamint a másodlagos címet küldő két rutin, melyek a LISTEN, illetve a TALK után adják ki a másodlagos címet (csatornakódot) a buszra.

IECOUT

A CHROUT rutin része. A CHROUT JMP-pal ugrik erre a rutinra, ha a készülék a soros buszon van.

A soros busz output egy 1 byte-os pufferen keresztül történik. A puffer címe $95, s egy flag ($94) jelzi, hogy a puffer üres-e. ($00 - üres; $80 - nem üres).

Kiírandó byte az akkumulátorban

```
IECOUT    BIT $94       ; puffer üres?
          BMI NEMURES
          SEC
          ROR $94
          BNE PUTBUF    ; feltétlen ugrás
NEMURES   PHA           ; byte mentés
          JSR $ED40     ; puffer kiírás a soros buszra
          PLA           ; byte vissza
PUTBUF    STA $95       ; byte a pufferbe
          CLC
          RTS
```

Készülékszám az akkumulátorban

```
TALK      ORA #$40      ; TALK kód
          .BYTE $2C
LISTEN    ORA #$20      ; LISTEN kód
          JSR $F0A4     ; A standard buszkezelés szempontjából üres
                          utasításnak tekinthető.
UENTRY    PHA
          BIT $94       ; puffer üres?
          BPL URES
          SEC
          ROR $A3       ; EOI (End Or Identify, közlemény utolsó byte-
                          ja) kód
          JSR $ED40     ; puffer kiírás
          LSR $94
          LSR $A3
URES      PLA
          STA $95       ;
          SEI
          JSR $EE97     ; DATA magas
          CMP #$3F      ; ATN magas?
          BNE NEM
          JSR $EE85     ; CLOCK magas
NEM       LDA $DD00     ;
          ORA #$08      ; ATN magas
          STA $DD00     ;
SENTRY    SEI
          JSR $EE8E     ; CLOCK alacsony
          JSR $EE97     ; DATA magas
          JSR $EEB3     ; cca. 1 ms várakozás
```

ezután következik a byte kiírása a pufferből a soros buszra, algoritmusát ld. később.

```
UNTALK    SEI
          JSR $EE8E     ;  CLOCK alacsony
          LDA $DD00
          ORA #$08      ;  ATN magas
          STA $DD00
          LDA #$5F      ;  UNTALK kód
          .BYTE $2C
UNLSTN    LDA #$3F      ;  UNLISTEN kód
          JSR UENTRY    ;  ld. fent
          JSR $EDBE     ;  ATN vissza
```

A továbbiakban a rutin 40 mikroszekundumot vár, majd a DATA és a CLOCK magasra állításával felszabadítja a buszt.

```
CHLIST    STA $95       ;  másodlagos cím LISTEN után. Az akkumulátor-
                           ban a másodlagos cím+96
          JSR SENTRY    ;  kiírás, ld. fent
```

A továbbiakban ATN vissza és RTS

```
CHTALK    STA $95       ;  másodlagos cím TALK után.
          JSR SENTRY
          SEI
          JSR $EEA0     ;  DATA alacsony
          JSR $EDBE     ;  ATN vissza
          JSR $EE85     ;  CLOCK magas
NEM       JSR $EEA9     ;  beszélő kész? (CLOCK alacsony?)
          BMI NEM
          CLI
          RTS
```

Az IECIN, byte olvasás a buszról és a $ED40, puffer kiírása a buszra rutinoknak csak az algoritmusát adjuk meg. Ebből az időzítések pontosan kiolvashatók és minden bizonnyal könnyebb követni, mint pl. a disassemblert.

Az algoritmust PASCAL nyelven írjuk le. Ennek az az oka, hogy az algoritmus bonyolult és keresni kellett a legtömörebb és legjobban áttekinthető megoldást. Az időzítési diagramok, bármennyire hasznosak is egyébként, azért nem tűntek megfelelőnek, mert a szignálok időbeli egymásutánja nem tükrözi kellően a kommunikáció algoritmusát, amelyben - az időzítésen kívül - egy kérdés-válasz rendszernek is szerepe van.

Olvasása azoknak sem fog nehézséget jelenteni, akik a PASCAL nyelvet nem ismerik.

Az algoritmusban használt nevek:

DATA-OUT, DATA-IN, CLOCK-OUT, CLOCK-IN: A soros busz vonalainak beállítása, ill. lekérdezése. A CIA 2 chip A portjának felső négy bitje. Az OUT vonalak invertáltak. a változók értéke a buszon megjelenő (logikai) feszültségszintnek felel meg.

WAIT(xx): Eljárás. xx időegységet várakozik. 1 időegység 1.042 mikroszekundum.

TIMER: Beállított értéke időegységenként automatikusan eggyel csökken.

ADATBIT[i]: Az adatbyte i-ik bitje.

EOI: End Or identify, a közlemény utolsó byte-ját jelző kapcsoló. Az író-rutinban paraméter (a hívó állítja be), az olvasó rutinban segédváltozó.

STATUS(s): Eljárás. A statusz változót ($90) állítja be.

Az időzítés áttekintésekor a PASCAL utasítások végrehajtási idejét nem kell figyelembe venni.

```
Procedure WRITEBYTE (EOI);
DATA-OUT:=1;
if DATA-IN=1 then begin STATUS=(128); stop end; "device not pr."
CLOCK-OUT:=1;
if EOI then begin
repeat until DATA-IN=1;"arra vár, hogy DATA-IN=1 teljesüljön"
repeat until DATA-IN=0 end;
repeat until DATA-IN=1;
CLOCK-OUT:=0;
for i:=7 downto 0 do begin
if DATA-IN=0 then begin STATUS(2); stop end; "timeout"
WAIT(56);
DATA-OUT:=ADATBIT[i];
CLOCK-OUT:=1;
WAIT(18);
DATA-OUT:=1; CLOCK-OUT:=0 end;
TIMER:=1024;
repeat until (DATA-IN=0) or (TIMER=0);
if TIMER=0 then STATUS(2); "timeout"
end;

Procedure IECIN;
EOI:=false;
CLOCK-OUT:=1;
repeat until CLOCK-IN=1;
KETYEG: TIMER:=256;
DATA-OUT:=1;
repeat until (CLOCK-IN=0) or (TIMER=0);
if TIMER=0 then
if EOI then begin STATUS(2); stop end
else begin
    EOI:=true;
    STATUS(64);
    DATA-OUT:=0;
    CLOCK-OUT:=1;
    WAIT(58);
    goto KETYEG end;
for i:=7 downto 0 do begin
repeat until CLOCK-IN=1;
ADATBIT[i]:=DATA-IN;
repeat until CLOCK-IN=0 end;
DATA-OUT:=0;
if EOI then begin
WAIT(40);
```

```
CLOCK-OUT:=1;
DATA-OUT:=1 end;
end;
```

Végül, a soros buszra vonatkozó programozási példaként megadjuk a CHKIN flow-ját, ha a készülék a soros busz. A CHKOUT-é ezzel analóg.

```
CHKIN   JSR $F30F     ; log. file keresése (file-szám a x regisz-
                        terben)
        BEQ OK
        JMP $F701     ; "file not open"
OK      JSR $F31F     ; file-paraméterek beállítása
        LDA $BA       ; készülék
        BEQ KEYBRD    ; billentyűzet
        CMP #3        ; képernyő?
        BEQ DISPLAY   ; ha igen
        BCS IECCHK    ; soros busz
        .
        .
IECCHK  TAX           ; készülék-szám mentés
        JSR $ED09     ; TALK
        LDA $B9       ; másodlagos cím
        BPL TOVABB    ; feltétlen ugrás
        .
        .
TOVABB  JSR $EDC7     ; másodlagos cím TALK után
        TXA           ; készülékszám vissza
        BIT $90       ; statusz
        BPL OK2
        JMP $F707     ; "device not present"
        .
        .
OK2     STA $99       ; output egység
        CLC           ; befejezési kód
        RTS
```

A C 64 MEGSZAKÍTÁSI RENDSZERE

A C 64 - hasonlóan más mikroprcesszoros rendszerekhez - kétféle megszakítást ismer, a maszkolható és a nem maszkolható megszakítást.

NMI - Non Maskable Interrupt
IRQ - Interrupt ReQuest

A C 64 rendszerben sokféle okból következhet be megszakitás. A megszakitáskérések közvetlenül négy forrásból futnak be az MPU (6510) chiphez. Ezek a következők:

- IRQ: CIA 1 chip-től
  a videokontroller (VIC) chip-től
- NMI: CIA 2 chip-től
  RESTORE billentyűtől.

Az alapsoftware működése közben a 15 lehetséges megszakítási ok közül csak néhány fordulhat elő, úgy mint:

- CIA 1 chip időzitőjének alulcsordulása (20 ms-onként)
- RESTORE billentyű leütése
- A cassette read vonal a CIA 1 chipen keresztül
- ha az RS232 vonal aktív, a CIA 2 időzítője szinkronizálja az átvitelt.

A CIA 1 időzítő okozta megszakítások állandóan működnek, ezért ezt a megszakítás rutint ismertetjük:

Négy funkciója van, ezek:

- a timer változó állítása
- a kurzor villogtatása
- a kazettás egység motorjának vezérlése
- a billentyűzet lekérdezése.

A két CIA chip - egymástól függetlenül - öt különböző okból generálhat megszakítást, úgy mint

- az "A" óra alulcsordulása
- a "B" óra alulcsordulása
- "Ébresztő" (ALARM)
- Adatbyte a soros kapun
- lefutó él a CIA chip FLAG lábán.

A CIA 1 FLAG lába a soros busz Service Request vonalára és a kazetta READ vonalára van kötve, a CIA 2 FLAG lába pedig a user portra.

A CIA 13-as ($0D) regisztere a megszakítás vezérlő regiszter.Egy adott megszakítást engedélyezhetünk a regiszter megfelelő bitjének 1-re állításával. Ebből a regiszterből olvasható ki a megszakítás oka is. Legmagasabb helyiértékű bitje jelzi, hogy történt-e megszakításkérés. Az olvasás hatására a regiszter nullázódik.

<u>Példa:</u>

Az alábbi - felhasználói - megszakítás rutin a RESTORE billentyű lenyomására feltétel nélkül megszakítja a futó BASIC programot úgy, hogy utána a CONT-tal folytatható. Hibakeresésnél igen hasznos lehet. (Ha pl. a program "lemerevedik" és STOP-pal nem lehet megszakítani.)

```
          *= $C800
          LDA #INTER<
          STA $318
          LDA #INTER>
          STA $319
          RTS
INTER     SEI
          PHA
          TXA
          PHA        ;  regiszterek mentése
          TYA
          PHA
          LDA #$7F
          STA $DD0D  ;  CIA 2 megszakítás tiltása
          LDA $DD0D  ;  megszakító regiszter olvasás
          BPL MINE   ;  CIA 2 okozta a megszakítást?
          JMP $FE72  ;  igen, ugrás az RS232 NMI rutinra
```

```
MINE     LDA #$7F    ; STOP flag
         STA $91     ; beállítása
         PLA
         TAY
         PLA         ; regiszterek visszatöltése
         TAX         ;
         PLA         ;
         RTI         ; ReTurn from Interrupt, visszatérés.
                     ; Megszakitás rutin csak így fejeződhet be.
                     ; Az RTS itt nem működne, úi. a megszakítás
                       hatására 3 byte kerül a stackbe; a vissza-
                       térési cím 2 byte-ján kívül a statuszre-
                       giszter értéke is.Megszakitás rutint emiatt
                       tilos JSR-rel hívni.
```

A rutin SYS51200-zal aktivizálható.

BASIC BŐVÍTÉSEK

BEVEZETÉS

A forgalomban lévő BASIC bővítésekkel is az I/O és a file struktúrák kialakításához nyujtott segítség szempontjából foglalkozunk.

Megjegyezzük, hogy a gép beépített BASIC interpreterével az adatkezelés, a fix mezőhossz és rekordhossz (tehát file-ok definiálása, kezelése) rendkívül nehézkes, lassú és nagy háttértár igényű. Így lényegében a beépített interpreter igényesebb adatfeldolgozó programok készítésére alkalmatlan. Ebben a fejezetben csak azokkal a software termékekkel foglalkozunk, melyek könnyen hozzáférhetők és a programozást segítik (nem helyettesítik). Tehát kész felhasználói rendszereket (pl. DATAMAT, FAKTUMAT, stb.), ill. adatbáziskezelő software-ket (pl. SUPERBASE, MASTER) nem tárgyalunk. Nem tárgyaljuk az egyébként igen hasznos HELP + programozást segítő software-t sem, mert ennek utasításai csak parancs módban működnek, az I/O programozására hatásuk nincs.

A bővítések tárgyalása előtt összefoglaljuk azokat a funkciókat,amelyre - az adatkezelés, adatfeldolgozás szempontjából - szükség volna. Az egyes funkcióknál utalunk rá, ha azt valamelyik bővítés - a tárgyaltak közül - tartalmazza. Ezután a bővítések I/O és adatszerkesztést támogató utasításait ismertetjük, majd összehasonlítjuk az egyes bővítéseket egymással.

A FONTOSABB BŐVÍTŐ FUNKCIÓK

Képernyő kezelés

Pozicionált PRINT utasítás (PRINT AT)
Ezt a funkciót a nagyobb számban nálunk elterjedt gépek BASIC interpretere beépítve tartalmazza (pl. Sinclair). A tárgyalt bővítések közül a SIMON'S BASIC és a DBASIC tartalmaz ilyen lehetőséget.

Kényelmesebb INPUT
Az adatfeldolgozó rendszerek mindegyikénél szükséges a kényelmes, lehetőleg ellenőrzött adatbevitelt lehetővé tevő INPUT utasítás. Ennek hiányában az állománykarbantartó (adatrögzítő) programok készítésénél az igényes és megfelelő sebességű adatbevitel megoldhatatlan.

A SIMON'S BASIC FETCH utasítása alkalmas a begépelés ellenőrzésére és a max. hossz definiálására.

A DBASIC INPUT[utasításánál kijelölhető a bekérés helye a képernyőn, definiálható a mező típusa (karakteres, valós, előjel nélküli egész), a bekérés hossza, kiolthható a kurzor, az input mező előtt és után is definiálható megjegyzés, stb.

HARDCOPY
A képernyőtartalom kiírása nyomtatón egy utasítással sokszor szükséges.

A C 64 PLUS tartalmaz hardcopy funkciót, de ez csak nagyfelbontású módban működik.

A SIMON'S BASIC-ben és a DBASIC-ben van kisfelbontású hardcopy. A DBASIC-nél a képernyő egy része is kinyomtatható.

Adatszerkesztés, adattárolás

String műveletek

A SIMON'S BASIC és a DBASIC is tartalmazza a legfontosabb stringkezelést megkönnyítő és meggyorsító műveleteket: beszúrás, rész felülírása, többszörözés, keresés. Mindkét bővítés tartalmaz numerikus értéket előkészítő utasítást nyomtatáshoz.

A DBASIC-ben van még stringcím lekérdező, stringtömbben kereső,stringtömb rendező utasítás is.

Aritmetika

Az aritmetikával kapcsolatban főként a különböző típusú értékek konverziója (dec<->bin, hexa<->dec) jelent segítséget, ha "már meg van írva".

Ilyen lehetőséget a SIMON'S BASIC és a DBASIC tartalmaz.

A SIMON'S BASIC-ben van más - sokszor hasznos - aritmetikai függvény is: osztási maradék képzés, exkluzív vagy, egész számok osztása.

Adattárolás

A C 64 rendszernél az adattárolás elég pazarló a háttértárolóknál (a számok között space is van, egy byte-ban csak egy számjegy tárolt, bináris szám nincs).

Ezen csak a DBASIC igyekszik segíteni. A numerikus értékek (egész és valós) "dupla sűrűséggel" tárolhatók (egy byte-ban két jel) stringként. Így a számok közötti space sem szükséges.

Memória kezelés

A C 64 BASIC interpreterének memória kezelése meglehetősen gyenge. 17 kbyte RAM BASIC-ből hozzá sem férhető. A hozzáférhető RAM-ot is csak byte-onként - igen lassan - lehet kezelni. Ráadásul a stringkezelés furcsa megoldása miatt (minden stringváltozó érték adásakor, sőt a stringgel végzett művelet munkaterületeként is újabb részt foglal le) a RAM hamar megtelik és következik a sokszor percekig tartó helyfelszabadító eljárás. Ez alatt a gép látszólag "merev". A bővítések közül csak a DBASIC próbálkozik a memória kezelés javításával. A stringkezelést teljesen átírja, a helyfelszabadító eljárás így elmarad. Lehetővé teszi a ROM-ok alatti RAM-ok kezelését, így a 17 kbyte felszabadul. Megkönnyíti néhány utasítással memória file-ok kezelését (lásd file-struktúrák).

I/O

Párhuzamos busz kezelése

Nagyobb háttértárak, nagyobb teljesítményű nyomtatók már nem ismerik az IEC buszt, párhuzamos az adatátvitel (IEEE-488). Forgalomban vannak olyan bővítések, melyek a párhuzamos busz kezelésén kívül mást "nem tudnak" (COMMODORE gyártmányú IEEE-488, hazai gyártású IEEE-488 SICOM).

A C 64 PLUS is tartalmaz IEEE-488 interface-t.

A vegyes buszkezelés mindegyik tárgyalt bővítésnél nehézkes. A soros - párhuzamos busz definiálását minden, már a másik buszra hivatkozó I/O utasításnál el kell végezni.

Tartalomjegyzék megjelenítése

Alapkövetelmény a diskette tartalomjegyzékének megjelenitése az aktuális BASIC program "megölése" nélkül.

Ilyen lehetőséget az IEEE-488 bővítésen kívül mindegyik tartalmaz a tárgyaltak közül.

I/O egyszerűsitett megadása

Értjük ez alatt szektor közvetlen behozatalát/kivitelét, a relatív rekordra pozicionálás egyszerű megadását, stb. Szektor I/O utasítást csak a DBASIC tartalmaz. Rekordra pozicionáló utasítást a DBASIC és a C 64 PLUS tartalmaz. (Az utóbbival nem megyünk semmire.)

A DBASIC-ben van még program módban is használható diszkette név és ID beolvasó utasítás is.

Használható INPUT# utasítás

A BASIC interpreter INPUT# utasítása a rekord szintű olvasást lehetetlenné teszi. (Erre a BASIC I/O utasításai fejezetben kitértünk.)

A tárgyalt bővítések közül a DBASIC tartalmaz olyan input utasítást, mellyel tetszőleges számú byte olvasható be egy nyitott file-ból, függetlenül a byte-ok tartalmától.

SIMON'S BASIC

A SIMON'S BASIC a standard BASIC terület elejéről "csíp le" 8 Kbyte-ot. Ez a veszteség megéri, mert cserébe egy olyan interpretert kapunk, amely sok, általában hasznos utasítással segíti a programozást. Tapasztalatunk, hogy mégis kevés helyen használják, aminek fő oka éppen a rengeteg új - és bonyolult szintaktikájú (egyéb utasítások 6-7 paramétert is tartalmaznak) - utasítás megtanulása.

Itt csak a könyvünk tárgykörébe eső utasításokat emeljük ki, a szintén jól használható hangkeltést elősegítő, ciklusszervező, stb. utasítást nem tárgyaljuk.

String műveletek

INSERT: string beszúrása egy másik stringbe

```
Szintaxis: INSERT "a string","b string",p
           a string : beszúrandó string
           b string : fogadó string
           p        : beszúrás kezdete a fogadó stringben
```

INST : string felülírása egy másik stringgel

```
Szintaxis: INST   "a string","b string",p
           a string : felülíró string
           b string : felülírandó string
           p        : felülírás kezdő pozíciója
```

PLACE : megadott string keresése egy másik stringben

Szintaxis: INST "a string","b string",p
a string : keresendő string
b string : string, amiben keresünk
p : az első megtalálás kezdő pozíciója b stringben (ha nincs benne p=0)

DUP : string többszörözése

Szintaxis: DUP "string",n

CENTRE: string sorközépre helyezése (képernyőre kerülő stringnél célszerű használni)

Szintaxis: CENTRE "string"

USE : numerikus érték szerkesztése

Szintaxis: USE "mask string","eredmény string":PRINT mask string: a mask tartalmazhat aktív maszkot és tetszőleges szöveget (pl. "ERTEK:####.##Ft")
eredmény string: a numerikus értéket tartalmazó string (pl. A$=STR$(A))

Képernyőkezelés

AT : kurzor pozicionálása PRINT paramétereként (a más gépeken megszokott utasítás)

Szintaxis: PRINT AT /s,o/ string

FETCH : ellenőrzött input

Szintaxis: FETCH "k.kar.",h,iv
k.kar.: kontrolkarakter(ek), mely az ellenőrzést definiálja:
HOME: csak nagy betűk
CRSR le: chr$(32)-chr$(64)
CRSR jobbra: nagybetűk és grafikus jelek
string: csak a string karakterei elfogadottak
h : az input jelek maximális száma
iv: input változó, amibe a begépelt jelek kerülnek

HRDCPY: a képernyő hardcopyja "lores" módban

FLASH : képernyőszín villogtatása

Szintaxis: FLASH sz,s
sz: villogtatandó szín kódja
s : villogás sebessége 1/16 sec.

OFF : FLASH kikapcsolása

BFLASH: képkeret színének váltogatása

Szintaxis: BFLASH s,sz1,sz2
s : változtatás sebessége
sz1: szín1 kódja
sz2: szín2 kódja

BFLASHO: BFLASH kikapcsolása

Aritmetika

MOD : osztási maradék képző függvény egész számoknál (ha a paraméterek nem egészek, azzá alakítja)

Szintaxis: MOD /x,y/
x: osztandó
y: osztó

DIV : egész számok osztása függvény
(ha a paraméterek nem egészek, azzá alakítja)

Szintaxis: DIV /x,y/
x: osztandó
y: osztó

FRAC : tizedes érték függvény

Szintaxis: FRAC /x/
x: szám, amelynek a tizedes részére van szükség

% : bináris szám konverziója decimálissá

Szintaxis: % bináris érték

$: hexadecimális szám konverziója decimálissá

Szintaxis: $ hexa jegyek

EXOR : exkluzív vagy két szám között (függvény)

Szintaxis: EXOR /x,y/

Diszkette kezelés

DISK : disk parancs kiadása

Szintaxis: DISK "parancs"
Az utasítás "megspórolja" a parancscsatorna megnyitását, lezárását, ill. a parancsot kiadó PRINT# utasítást.

DIR : directory, ill. a directory egy részének listázása

Szintaxis: DIR "$: j
j: joker (* vagy ?), lásd pl. SCRATCH parancs

COMMODORE IEEE-488

Az IEEE-488 új utasításokat nem tartalmaz, feladata a párhuzamos busz kezelése. Működése a felhasználó számára "észrevétlen", a standard I/O utasításokkal kezelheti a párhuzamos buszt. A hozzáférhető IEEE-488 buszkezelő bővítés a gép modul portjára csatlakoztatható. A bővítés ROM-ban van, mely bekapcsoláskor átmenti magát $C830 - $CFFF-ig a RAM-ba. Az átmentés után a ROM kikapcsolódik. (A BASIC terület nem csökken.)

A soros és párhuzamos busz közötti átkapcsolás SYS utasítással történik:

SYS64789 : soros busz kiválasztása
SYS51364 : párhuzamos busz kiválasztása

A STOP-RESTORE megnyomása a soros buszkijelölést visszaállítja.

Ha a párhuzamos buszt egyedileg akarjuk kezelni, esetleg megváltoztatni a kezelést, erre a RAM-ban lévő rutinok lehetőséget adnak.

C 64 PLUS

A bővítést a Roland Kohler Gesmbh gyártja (Ausztria). A modul portra csatlakoztatható bővítés elvégzi a párhuzamos busz kezelését, tartalmazza a BASIC 4.0 DOS verziót, tartalmaz assembler monitort és van benne néhány nagyfelbontású grafikát kezelő utasítás. Itt csak az I/O utasítások közül néhánnyal foglalkozunk (a DOS 4.0 leírását lásd a fejezet végén). A modul a párhuzamos busz kezelésén kívül is sok programozást segítő utasítást tartalmaz, cserébe a BASIC területből elvesz 8 kbyte-ot.

DS,DS$: a diszk státuszának fenntartott változók

Igen jó lehetőség, az I/O utasítások után az esetleges hiba megállapitására a parancscsatorna olvasása nélkül. (A hiba megállapitásra ez az egyetlen lehetőség, mert a hiba fellépeskor a modul által kiadott BASIC üzenet és a hiba között gyakran nincs semmilyen összefüggés. Sajnos a fenntartott változók kezelését is elég nehézkesen oldották meg, jelentősen lassul az utasítás végrehajtás.)

IOSELn: a használni kívánt busz kijelölése
- n=1: soros busz
- n=0: párhuzamos busz
- n=2: user porton használt 4-es készülékszámú nyomtató jelzése (ha a készülékszám más, a megadás: IOSEL2,k)

A user porton használható nyomtató típusa POKE708,t utasítással állítható be. Az alapértelmezés 128 (GP 700A).

- t=0: olyan nyomtató, melyre az ASCII kódok változtatás nélkül küldhetők ki (EPSON, SEIKOSHA)
- t=128: a különleges jeleket a C 64 jelgenerátor dekódolja
- t=240: mint a 128, de szín kezelés nélkül (EPSON FX 80, EPSON RX80)

DIRECTORY: tartalomjegyzék betöltése
DIRECTORY [Dm] [Uk]
m: meghajtó száma
k: készülékszám

A tartalomjegyzéket a BASIC program sérülése nélkül tölti be. Csak párhuzamos busz esetén használható. Funkciójában ugyanaz, mint a CATALOG, csak sokkal gyorsabb.

RECORD : pozicionálás relatív rekordra
RECORD#lf,rs,p
lf: logikai file-szám
rs: rekord sorszáma
p : pozíció (elhagyható paraméter)

Az rs paraméter változóként nem adható meg, csak numerikus konstansként. (az utasítás csak program módban működik. Egy program futása közben rs értéke konstansként általában nem ismert. Így az utasítás lényegében használhatatlan.)

DBASIC

A DBASIC adatfeldolgozási célokra kifejlesztett interpreter, a BASIC területén kívül ($C000-$CFFF) működik. Általában megnöveli a gép erőforrásait, hatékonyságát. (Elérhetővé teszi basic-ból a ROM-ok alatti RAM-okat, pakolt számok használatával megnő a diszkette kapacitása, a standard interpreterben elég szerencsétlenül megoldott string kezelés átírásával elmarad a memória gyakori megtelése "string szeméttel").

A DBASIC hazai készítésű, könnyen hozzáférhető, így csak az utasításokat írjuk le, a szintaktika leírásától eltekintünk.

Diszkette I/O

INPUT@ : diszkette szektor közvetlen olvasása
PRINT@ : diszkette szektor közvetlen írása
INPUT* : meghatározott számú byte olvasása nyitott file-ból
LOAD$: diszkette tartalomjegyzék beolvasása
LOAD* : diszkette név és ID beolvasása
PRINT#[: pozicionálás relatív rekordra

Képernyő kezelés

PRINT[: kurzor pozicionálása PRINT paraméterként (AT)
PRINT£ : hardcopy, a kinyomtatott sorok beállításával
INPUT[: szerkesztett, ellenőrzött input

Memória kezelés

LOAD[: RAM részének áthelyezése
PEEK : RAM olvasása
CLR* : verem törlése
CLR# : tömbök törlése

Program elágazások

IF : then nélkül a következő utasítás az else ág
GOTO, GOSUB, RESTORE: kiszámítható sorszám
x f$: feltétel kiértékelés stringként
ONERROR: hibakezelés programon belül
CLRERROR: hibakezelés törlése

String műveletek

String műveletek:
dump, cím lekérdezés, string kifejezés keresése string kifejezésben; string tömbben; memóriában, beszúrás stringbe, rész cseréje, string tömb rendezése.

Aritmetika

Aritmetikai műveletek:
aritmetikai érték szerkesztése maszkkal, stringben adott aritmetikai kifejezés kiértékelése. numerikus érték tömörítése, visszaalakítása.

DOS 4.0

A BASIC bővítések egy része, ill. a későbbi fejlesztésű COMMODORE gépek már a BASIC interpreterben is tartalmazzák a DOS 4.0 verziót. Természetesen ez semmi különbséget nem jelent a floppynak küldött parancs szempontjából, de egyes utasítások szintaktikája kényelmesebb. Itt azokat az utasításokat írjuk le, melyek eltérnek a C 64 interpreter által kezelt utasításoktól. Az utasításoknál a [] jelek közé zárt paraméterek megadása nem kötelező, van alapértelmezés szerinti értékük. Az alapértelmezések:

meghajtó száma: 0 (ill. az utolsó hivatkozás)
készülékszám : 8
file
megnyitási mód: R (olvasás)

Az utasítások:

APPEND : szekvenciális file bővítése a file jelenlegi végétől

APPEND#lf,"file név"[,m] [Uk]
lf: logikai file-név
m: meghajtó száma
k: készülék száma

BACKUP : diszkette másolása (csak a két meghajtós floppykon van értelmezve)
BACKUP Dm1 TO Dm2 [Uk]
m1: meghajtó, amin a forrás lemez van
M2: meghajtó, amin a másolat lesz
k: készülékszám

CATALOG: lemez katalógus olvasása és képernyőn megjelenítése (a BASIC terület változatlan marad)

```
CATALOG [Dm] [Uk]
        m: meghajtó száma
        k: készülékszám

COLLECT: pontosan megegyezik a validate utasítással
COLLECT Dm [Uk]
        m: meghajtó szám
        k: készülékszám

CONCAT: két szekvenciális file összekapcsolása
CONCAT [Dm1],"f1" TO [Dm2] "f2" [Uk]
       m1: az első file meghajtó száma
       m2: a második file meghajtó száma
       f1: az első file neve
       f2: a második file neve
        k: készülékszám

DCLOSE: file(ok) lezárása
        ha nincs logikai file-szám megadva, a  kijelölt  készüléken  minden
        file-t lezár
DCLOSE [#lf] [Uk]
       lf: logikai file-szám
        k: készülékszám

DLOAD : program file betöltése
        hatása azonos a load utasítással
DLOAD [Dm] [Uk],"fn"
        m: meghajtó száma
        k: készülékszám
       fn: file-név

DOPEN : file megnyitása
DOPEN#lf,"fn" [,t] [,Dm] [Uk]
       lf: logikai file-szám
       fn: file-név
        m: meghajtó száma
        k: készülékszám
        t: file típus - megnyitási mód
           t=W: szekvenciális file írása
           t=R: szekvenciális file olvasása
           t=L: relatív file

DSAVE : program mentése
        hatása azonos a save utasítással
DSAVE [Dm] [Uk],"fn"
        m: meghajtó száma
        k: készülékszám
       fn: file-név

HEADER: diszkette formázása
        hatása azonos a new utasítással
HEADER"dn" [,Dm] [Uk] [,Iid]
       dn: diszkette neve
        m: meghajtó száma
        k: készülékszám
       id: diszkette ID
```

RECORD: pozicionálás relatív rekordra
hatása azonos a P utasítással

RECORD#lf,rs[,bp]
lf: logikai file-szám
rs: rekord sorszám
bp: byte pozíció a rekordban

RENAME: file-név átnevezése
hatása azonos a rename utasítással

RENAME [Dm] [Uk],"rf" TO "uf"
m: meghajtó száma
k: készülékszám
rf: régi file-név
uf: új file-név

SCRATCH: file törlése

SCRATCH [Dm] [Uk],"fn"
m: meghajtó száma
k: készülékszám
fn: file-név

A direkt módban kiadott utasítás után megjelenik az
ARE YOU SURE?
kérdés (biztos benne?), ha a válasz "Y", a törlés megtörténik, egyébként nem történik semmi.

BŐVÍTÉSEK ÖSSZEHASONLÍTÁSA

SIMON'S BASIC: Nagyon sok új utasítás van benne (több, mint 100), ezek legnagyobb része jó. A definiált utasítások készítői talán - az elég gyenge - C 64 BASIC interpreter kiegészítésének szánták. Ezt a feladatot be is tölti. Olyan munkánál célszerű használni, ahol grafika, hanggenerálás, numerikus műveletek együtt jelentkeznek. 12 Kbyte-ot foglal el (8-at a BASIC területből és a $C000-$CFFF-ig).

Hiányossága, hogy sem assembler monitor, sem más programírás közben szükséges funkció (pl. HELP + #A, #B, #F, #C, stb.) nincs benne és nem is tud más bővítésekkel együtt működni.

COMMODORE IEEE-488: Az elvárás a párhuzamos busz kezelése, ezt teljesíti. A BASIC területet nem csökkenti, a szabad RAM-ban dolgozik 2 Kbyte-on ($C800-$CFFF).

Programozást segítő más bővítés (pl. HELP +) betölthető mellé.

C 64 PLUS: Van benne egy kevés nagyfelbontású grafika (majdnem semmi). Tartalmazza a DOS 4.0 verziót. Ez eddig nem sokat segít. Ami nagyon hasznos: a párhuzamos busz kezelése, assembler monitor és a HELP + "#" parancsaival megegyező funkciók (itt "f" a nevük).

12 Kbyte a működési területe (8 Kbyte a BASIC területből és a $C000-$CFFF).

A C 64 PLUS lényegében egyesíti a HELP + fontosabb funkcióit és a párhuzamos buszkezelést.

DBASIC: Az egyetlen olyan bővítés, ami adatfeldolgozás, I/O, file-kezelés támogatására készült. Ezt a feladatot jól el is látja.

A BASIC területet nem csökkenti, a szabad RAM 4 Kbyte-ján működik ($C000-$CFFF).

A definiált új utasítások parancs módban is működnek (az INPUT is), ez a tesztelést, a diszkette tartalom vizsgálatát megkönnyiti.

Ami hiányzik belőle: a párhuzamos busz kezelése és a programozást segítő funkciók (lényegében a HELP +). A hiányok pótolhatók, a DBASIC más bővítésekkel együtt is működik (pl. HELP +, C 64 PLUS, stb.).

FILE STRUKTÚRÁK

BEVEZETÉS

Olvasóink közül azok, akik nem a C 64-en keresztül ismerkedtek meg a számítástechnikával, más "komolyabb" gépen már van gyakorlatuk, a C 64 adatkezelését, a file típusokat nem találják kielégítőnek. (A gépet nem is erre tervezték. Egy játékprogram betöltéséhez, esetleg a házi könyvtár katalógusának elkészítéséhez a beépített basic interpreter elegendő.)

Nálunk a gépet - olcsósága miatt - a vállalati ügyvitelben, adatfeldolgozási célokra használják. És erre meg van a lehetőség. A beépített hardware, a géphez csatlakoztatható perifériák teszik a gépet "profi" géppé. (Annak ellenére, hogy a gép ára kb. a háromszorosa a "nyugati" árnak, mégis a legelterjedtebb a maga (ár)kategóriájában. Ezen az sem tudott változtatni, hogy a magyar számítástechnikai vállalatok által gyártott gépek a bekerülési költséghez közeli áron kerülnek forgalomba, az árat megháromszorozó "üzletpolitika" őket nem sújtja.)

Tehát a hardware jó, a software-en igyekszünk segíteni!

A C 64 file típusait részletesen nem tárgyaljuk, ez minden C 64 és floppy kézikönyvben benne van. Olyan adatszerkezetek felépítésére és karbantartására térünk ki, melyek a számítástechnikában bevezetettek, de megvalósításuk a C 64 gépen nem elterjedt.

Kiinduló típusként relatív file-t fogunk használni. Ez az egy típus elegendő is. A megvalósítható logikai adatszerkezetek tárgyalása előtt kitérünk általános adatkezelési problémákra, a relatív file - mint kiinduló típus - néhány jellegzetességére és a memóriában létrehozható indexfileok - mint segédfile-ok - tárgyalására. Az egyes file, ill. adatszerkezetek tárgyalásánál a három előzetes fejezetben foglaltakat ismertnek tételezzük fel.

A C 64 ADATKEZELÉSE

Ebben a fejezetben áttekintjük, hogy milyen fogalmak és megoldandó problémák vannak. Mik a C 64 lehetőségei, hogyan bővíthetjük ki a lehetőségeket?

Adatmező a rekordban

Az adatmező egy olyan változó - a rekord részeként - melynek típusa, hossza definiált. A C 64 az I/O szempontjából lényegében egy típust ismer, a stringet (egy a rekordba kiírt numerikus változó is stringként kerül tárolásra). A hossz nem definiálható, a változó aktuális értékétől függ.

A nehézségek már az adat megjelenésével kezdődnek, vagyis a képernyőn bekéréssel. A beépített INPUT utasítás a bekérésre alkalmatlan. Ha egy szerkesztett képernyőn az INPUT utasítás után pl. a kurzorral elmegyünk egy másik sorra és ott gépeljük be a mezőt, a return után az input változóban annak a sornak a tartalmát kapjuk, ahol a kurzor éppen állt. BASIC-ben két - lényegében azonos - megoldás van.

Az egyik a GET használata. A kurzort a megfelelő helyre pozicionáljuk, az input mező karaktereit egyenként bekérjük és visszaírjuk.

A másik a billentyű puffer olvasása (a kurzor pozicionálás után):

```
WAIT 198,1
GET X$
```

A bekért karaktert most is vissza kell írnunk. Mindkét esetben gondoskodnunk kell a karakter ellenőrzéséről, a kívánt adattípusnak megfelelően.

Következik a mező hosszának beállítása. Ha a mező string, vagy numerikus egész típusú, csak töltő zerót, ill.space-t kell hozzáadnunk balról, ill. jobbról.

Ha string:

```
I$=LEFT$(I$+LEFT$(S$,H),H)
H : a kívánt hossz
S$: space-eket tartalmaz
```

Ha numerikus egész:

```
I$=RIGHT$(LEFT$(Z$,+I$,H)
H : a kívánt hossz
Z$: Z$: zerókat tartalmaz
```

Ha a mező előjeles egész, az előjelről külön kell gondoskodnunk:

```
E$=LEFT$(STR$(VAL(I$)),1)
I$=E$+RIGHT$(LEFT$(Z$,H-1)+MID$(I$,2.LEN(I$)-1),H-1)
E$: előjel
```

Ha a mező valós, az eljárás még nehézkesebb. A mező bekérésre és hossz beállításra célszerű assembler rutint írnunk. A mező ellenőrzött bekérésére használhatjuk a SIMON'S BASIC FETCH utasítását, vagy a DBASIC INPUT[utasítását (ez elvégzi a hossz beállítását és a kurzor pozicionálást is).

A numerikus értékek tárolásához - a diszkett kapacitás megnövelésére - használhatunk tömörebb formát is. A legtömörebb a bináris tárolás. Így csak egész értékeket tárolhatunk. Három bináris byte-nál nagyobb értékeknél a konverzió már észrevehető idejű.

A konverzió BASIC-ben (3 byte-ra):

```
X : VAL(I$)
X$: a konvertált  érték
X1%=X/65536
X2%=(X-X1%*65536)/256
X3%=X-X1%*65536-X2%*256
X$=CHR$(X1%)+CHR$(X2%)+CHR$(X3%)
```

A vissza konvertálás:

```
X=ASC(LEFT$(X$,1))*65536+ASC(MID$(X$,2,1)*256+ASC(RIGHT$(X$,1))
```

Másik lehetőség a tömörítésre a számok "pakolt" ábrázolása (egy byte-ban két jegy).

Egy megoldás BASIC-ben páros jegyű egész számok pakolására:

```
X$=""
FOR I=0 TO LEN(I$)-1 STEP 2
X1=ASC(MID$(I$,I+1,1))-48
X2=ASC(MID$(I$,I+2,1))-48
X$=X$+CHR$(X1*10+X2)
NEXT
```

A vissza pakolás:

```
X$=""
FOR I=1 TO LEN(I$)
X=ASC(MID$(I$,I,1))
X1%=X/10
X2%=X-X1%*10
X$=X$+CHR$(X1%+48)+CHR$(X2%+48)
NEXT
```

A konverziókhoz is célszerű assembler rutint írnunk. A DBASIC LET*P - LET*U utasítás párjával valós számokat is használhatunk sűrítve.

Rekord I/O

A rekord kötött hosszúságú, típusú, együtt kezelt adatmezőkből áll. Az együtt kezelt alatt azt értjük, hogy a rekordot egy I/O utasítással szeretnénk mozgatni.

Ha a mezők hossza állandó, a rekord hossza adódik, csak össze kell adni a mezőket kiírás előtt!

```
O$=M1$+M2$+....Mn$
```

A kiírás így egy utasítás:

```
PRINT#1f,O$
```

A rekord beolvasása már nehezebb. BASIC-ben nem is lehetséges. (Az INPUT#1f,I$ csak a rekordban levő első szóközig olvas.)

A megoldás szintén BASIC bővítés használata, vagy saját assembler rutin írása. (Erre az INPUT# utasítás leírásánál adtunk példát.)

A C 64 file-ok

A file rekordokból áll. A rekordokat szeretnénk karbantartani. A file méretét a rekordok száma ismeretében jó lenne előre tudni.

Ha adatmezőink hossza nem állandó, a rekordokat egy I/O utasítással mozgatni nem tudjuk, a karbantartás nehézkes, a file méretét nem tudjuk előre, mert a rekordhossz változik.

A bevezetésben említettük, hogy részletesen nem foglalkozunk a C 64-en definiált file típusokkal. Nézzük meg röviden a definiált file típusokat, miért nem fogjuk használni őket igényesebb feladatoknál.

A legegyszerűbb a szekvenciális (soros) file. A rekordok közvetlen elérése nem lehetséges, egy rekord karbantartásához az egész file-t újra kell írni.

(Láttunk működni egy rendszert, ami egy törzsállomány karbantartását végezte. A rendszer számtalan diszkettet használt. A törzsállomány egy teljes diszketten fért el. A feldolgozás egy fél diszkettnyi különböző típusú tranzakciós rekorddal tartotta karban a törzsfile-t. Reggel nyolckor megjelent a képernyőn: TURELMET KEREK, DOLGOZOM! Ezután már csak a diszketteket kellett cserélgetni időnként. Délután 3-kor "lelőttük" a programot.)

A relatív file az egyetlen, a gyakorlatban is hasznosítható file típus. Közvetlen elérésről itt is csak akkor lehet szó, ha a rekordok azono-

sítására megfelel a folyamatos sorszám. Vagy találunk olyan algoritmust, amivel a rekordok azonosítói (kulcsa) egy sorszám intervallumra egyértelműen lekérdezhetők, lehetőleg "hézagok" nélkül. Ez általában nem szokott sikerülni.

Hátra van még - a kézikönyvekben a haladóknak ajánlott - direkt file. Itt a helyzet hasonló a relatív file-hoz. Az a különbség, hogy azt a táblázatot, ami a rekord helyét a diszketten kijelöli, szintén nekünk kell kitalálnunk és kezelnünk. Gondoskodnunk kell a kijelölt szektor lefoglalásáról, ill. felszabadításáról a BAM-ban. További hátrányként jelentkezik, hogy a felhasznált szektorok nem eléggé védettek. Egy véletlenül kiadott VALIDATE parancs a file-t megsemmisiti. (Ha programokat is tárolunk ugyanazon a diszketten, erre időnként szükség lehet.)

Mégegyszer megemlítjük, hogy a fenti problémák főként adatfeldolgozási feladatoknál jelentkeznek. Kis I/O igényű, számítás igényes feladatok a beépített interpreterrel - felületes gépismerettel is - megoldhatók.

Ha igényesebb adatfeldolgozási feladatokat kívánunk megoldani, a géppel való játszás és az "éles" munka közé célszerű még egy fázist beépíteni. Nevezetesen valamelyik adatkezelésre alkalmas bővítés beszerzését és begyakorlását, vagy a szükséges rutinok megírását assemblerben.

RELATÍV FILE

A relatív file egy olyan speciális ISAM file, ahol a rekord kulcsa (indexe) csak folyamatos sorszám lehet és a kulcs nem része a rekordnak, az indextábla egyszintű. Ezzel a megkötéssel az ISAM file-ok használatának szokásos lehetőségei fennállnak:

- A file az elejéről, vagy egy adott kulcstól kezdve szekvenciálisan olvasható, írható. A file megnyitása után, pozicionáló utasítás nélkül a mutató az első rekordra áll. Ha egy pozicionáló utasítás után pozicionáló utasítás nélkül olvassuk/írjuk a file-t, a feldolgozás szekvenciális lesz.

- Bármely rekord közvetlenül olvasható/írható a kulcsa (a rekord relatív sorszáma) alapján.

Azt állítottuk, hogy a relatív file alapján minden - egyáltalán lehetséges - struktúra megvalósítható. Ez fennáll a C 64 két másik file típusára is.

A szekvenciális feldolgozást a relatív szerkezet lehetővé teszi. Pozicionálás nélkül az elejétől olvassuk a file-t, a file végét az első olyan rekord jelenti, melynek beolvasásakor az első byte értéke $FF.

A direct feldolgozás helyettesíthető olyan relatív file létrehozásával, ahol a rekord hossza egy szektornyi (csak 253 byte, mert a szektor első két byte-ja a következő szektor mutatója,az utolsó byte pedig a carriage return). A feldolgozás egyszerűbb is, mert a szektorok lefoglalására, felszabadítására, saját indextábla kezelésére nincs szükség. A file véletlenül nem semmisülhet meg.

INDEX FILE-OK A MEMÓRIÁBAN

Annak érdekében, hogy tényleg jól használható adatszerkezeteket,file struktúrákat tudjunk kialakítani, ki kell térnünk még a memóriára, mint a file-ok egy lehetséges tárolójára.

A relativ file-nál láttuk, hogy speciális ISAM file-ként viselkedik. A legtöbb gyakorlati feladatnál "igazi" kulcsra van szükség, ami tetszőleges típusú adatmező lehet a rekordban (vagy a rekordon kívül). Ha pl. a kulcs a rekordban van és relatív file-t használunk - a szekvenciális feldolgozáshoz hasonlóan - az adott kulcsú rekord megkereséséhez addig kellene a file-t olvasnunk, amíg beolvassuk azt a rekordot, amelyben a kulcs van. Ez az eljárás használhatatlan, mert átlagosan a file felének beolvasását jelenti. A keresés meggyorsítására segédfile-t hozunk létre, amit állandóan a memóriában tartunk. Ez az indexfile. Az indexfile rekordja általában két részből (mezőből) áll:

- az adatrekord kulcsa (azonosítója)
- sorszám (az adatrekord relatív sorszáma), ami az adatrekord helyét jelöli ki az adatfile-ban.

Az indexfile memóriában tárolására két lehetőség van:

- Az index rekordokat tömbelemként tároljuk (stringtömb), a tömbelem mindkét mezőt tartalmazza, pl.:

cikkszám rs

Cikkszám: egy cikk azonosítója egy cikktörzsben
rs: a rekordsorszám a cikktörzsben

- Az index rekordokat a BASIC területén kívül, a RAM-ban tároljuk.

Egy rekord ugyanolyan, mint az előbbi tömbelem. Az indexfile-t a floppyn tárolhatjuk pl. relatív file-ban, de tárolhatjuk programfile-ként is (a beolvasáshoz és kivitelhez használhatjuk az I/O rutinok között a LOAD és a SAVE-nél adott példa rutinokat), ill. szekvenciális file-ként.

A tömbkénti tárolás azzal az előnnyel jár, hogy a rekordokat BASIC utasításokkal is kezelni tudjuk. A hátrány a memória rossz kihasználása. Egyrészt csak a BASIC területét használhatjuk, másrészt minden rekord 3 byte-al hosszabb a szükségesnél. (A BASIC minden string tárolásához 3 plusz byte-ot felhasznál. Az első byte a string hossza, a 2-3. byte a címe.) E mellett a tömbelemeknek a programban gyakorta adunk új értéket, a memória hamar megtelik és következik a szemétgyűjtő eljárás, miközben a gép "merev".

A BASIC területén kívüli tárolás egyetlen hátránya, hogy BASIC utasításokkal nem tudjuk a rekordokat kezelni. Egyrészt a ROM-ok "alatti" RAM-okba csak írni tudunk (POKE), olvasni nem tudunk (ha a RAM "felett" ROM van, a PEEK a ROM-ot olvassa). Másrészt, ha biztosított lenne nagyobb RAM terület (pl. a BASIC terület végét "lehuzzuk"), a rekordok byte-onkénti kezelése a PEEK, POKE segítségével elég lassú.

Az indexfile mérete, az elhelyezhető rekordok száma, a választott elhelyezési módszertől függ. A BASIC terület mérete 38 kbyte. Ha feltesszük, hogy a karbantartó program (a szükséges változó területekkel együtt) 12 kbyte, marad 26 kbyte.

Példa:

Legyen a kulcs hossza 8 byte, ez a relatív rekordsorszámmal együtt - melyet binárisan tárolunk - 10 byte. Ha a tömbbeni tárolást választjuk, 2048 index fér el (26*1024/13). Ha a rekordokat közvetlenül a RAM-ban tároljuk, lényegesen több index fér el. A BASIC területét "lehuzzuk" 26 kbyte-al (POKE56, 56:CLR). Assemblerben fogunk dolgozni, a ROM-ok "alatti" RAM is használható

lesz. A $C000-$CFFF-ig terjedő területét fenntartjuk az assembler rutinoknak. Így kapunk egy egybefüggő területet a BASIC végétől, ez 34 kbyte ($3900-$BFFF-ig). Itt 3481 rekord tárolható (34*1024/10). Rendelkezésünkre áll még a KERNAL "alatti" RAM ($E000-$FFFF-ig), ez 8 kbyte. Itt is tárolhatunk 819 rekordot (8*1024/10). Így összesen 4300 rekord tárolható.

Vegyünk a fentiekre még egy gyakorlati példát:

A feladat: vállalati készáru raktár kezelése. Az áru megjelölő adatait és a készletadatokat egy rekordban tároljuk. A rekord hossza így (pl.) 100 byte-ra adódik. A vállalat sok féle terméket gyárt. A termékek azonosítására 7-jegyű termék számot használnak. Max. hány termékből álló választékot tudunk kezelni?

A 7-jegyű kulcs tárolására használhatunk 3 byte-os bináris számot. Az indexrekord így 5 byte hosszú. Az indexfile-t tároljuk a RAM-ban. Így 8601 (42*1024/5) rekordot tudunk tárolni.

Bár nem tartozik szorosan ide, nézzük meg, milyen háttértárolót célszerű választani. Az adatfile és a kulcsfile relatív file lesz. A háttértár igény:

```
 8601*6      (a 6. byte a carriage return)
+8601*101    (a 101. byte a carriage return)
_________
 920307
```

Ez 3624 szektort jelent (920307/254), plusz az oldalszektorok. Célszerű COMMODORE SFD1001 floppyt alkalmazni. Ha a vállalat pl. csak 1200 terméket gyárt, elegendő a COMMODORE 1541-es floppy. Az indexfile-t így tömbben is tárolhatjuk (még a 7-jegyű termékszámot közvetlenül kulcsként használva is).

Tekintsük át, hogy a memória rekordok kezelésénél milyen feladatok merülnek fel. A feladatok egy része elvégezhető BASIC-ben, de többnyire assembler rutinokat kell írnunk. (A forgalomban lévő bővítések közül csak a DBASIC tartalmaz a feladatokhoz kapcsolódó, már "megírt" utasításokat.)

Indexrekord olvasása

Az olvasás elsősorban egy adott index megkeresését jelenti az indexfileban.

Tömb használata esetén a módszer attól függhet, rendezett-e az indexfile a kulcsra. Ha igen, BASIC-ben is kivárható idő (0.2-0.5 sec) alatt hozzáférhetünk a kulcshoz pl. bináris kereséssel. Ha a tömb rendezetlen, a keresés szekvenciálisan 10 sec-ig is eltarthat.

Itt is csak assembler rutin segíthet (DBASIC: LE#F).

Ha a file RAM-ban van, mindenképpen assembler rutint kell írnunk (DBASIC: LET#M).

Indexrekord írása

Tömb használata esetén ez egyszerű. Ha egy rekord módosításáról van szó, a tömbindexet már ismerjük (előzőleg meg kellett állapítanunk), így ez csak egy értékadás. Ha a file bővítése miatt írunk, az első üres tömbelemet kell megkeresnünk és ebbe írjuk az indexrekordot.

Ha a file RAM-ban van és módosításról van szó, a rekord helyét már megállapítottuk (ellenőriztük, hogy van-e ilyen kulcsú rekord). Ha bővítés miatt írunk, "üres" helyet kell keresnünk. Ez sem nehéz, ha nyilvántartjuk, hogy az előző bővítő rekordot hova írtuk, mert az ez utáni byte-okba kell most írnunk. A beírandó indexrekord lehet egy string-változóban, vagy lehet a memóriában az erre fenntartott helyen.

Ha string-változóban állítottuk össze a rekordot, ismernünk kell a string címét. A cím megállapításhoz assembler rutint kell írnunk (DBASIC: LET*A). A rekord beírását végezhetjük PEEK és POKE segítségével (lassú), vagy újabb assembler rutint kell írnunk (DBASIC: LOAD[).

Rendezés

Ha a kulcsok keresését BASIC-ben szeretnénk elintézni (lásd fent),a string-tömbnek - ami az indexfile-t tartalmazza - rendezettnek kell lennie. A tömb rendezése (kivárható idő alatt) BASIC-ben nem végezhető el, a rendezést assemblerben kell megírnunk (DBASIC: LET#S).

Speciális indexfile-ok

Itt speciális alatt azt értjük, hogy az indexrekord felépítése eltér a korábban megadottól (kulcs + relatív rekord sorszám).

Ha a korábban megadott rekord felépítést használjuk, az indexfile alapján biztosított az adatrekordok közvetlen elérése. Ha az indexfile a kulcsra rendezett, az adatrekordokat a kulcsra rendezve is elérhetjük. (Indexszekvenciális feldolgozás.)

Ha az adatrekordok kulcs szerint rendezett soros elérése nem követelmény, csak a közvetlen hozzáférés, az indexrekordból az adatrekord relatív sorszámát elhagyhatjuk. Új indexrekord az adatfile bővülésekor keletkezik. Így kereséskor a megtalált kulcs helye az indexfile-ban kijelöli az adatfile-ban a relatív rekordsorszámot. A sorszámot az adja, hogy hányadikként találtuk meg a keresett kulcsot az indexfile-ban a kulcs között. Ez a módszer - ha a kulcs kevés byte-ból áll - lényegesen javítja a memória helykihasználását.

Térjünk vissza a korábban említett példára. A kulcs 7-jegyű egész szám volt, melyet binárisan ábrázoltunk 3 byte-ban. Az indexrekord hossza a relatív sorszámmal együtt 5 byte volt. Így 8601 indexet tudtunk tárolni. Ha a relatív sorszámot elhagyjuk, 14335 kulcs tárolható.

Eddig olyan szerkezeteket tárgyaltunk, ahol az indexfile egy adatfilera mutatott. Bonyolultabb esetben az indexfile két (vagy több) adatfile kapcsolatát is leírhatja a - mindegyik file-ban azonos - kulcs alapján.

Vegyük elő a korábbi példát, a készáru raktár kezelését. A megjelölő adatokat és a készletadatokat egy állományban tároltuk.

A raktárforgalom során keletkező "mozgás" rekordokról nem volt szó. Ha ezeket pl. lista szerkezetben tároljuk (lásd a következő fejezetben), az indexrekordban elhelyezhetjük a lista fejének mutatóját (annak a rekordnak a relatív sorszámát, amelyik a lista kezdete). Így az indexrekord hossza 2 byte-val megnő:

cikkszám sc mf

cikkszám: cikk azonositója
sc: a cikk azonosítóhoz tartozó relatív sorszám, a készletfile-ban a cikkhez tartozó rekord relatív sorszáma
mf: a lista szerkezetű forgalom file-ban a cikkhez tartozó forgalmi rekordok listájának a fejére (a lista első elemére) mutató relatív rekord sorszám

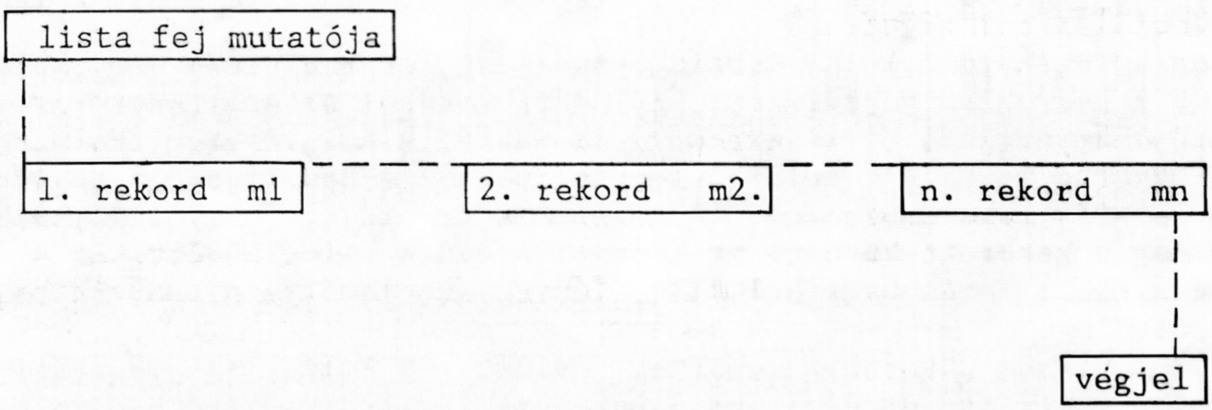
Az indexrekord hosszának növekedéséért cserében, arra a kérdésre, hogy egy adott termék raktári mozgásának tételei melyek voltak az elmúlt időszakban, anélkül tudunk válaszolni, hogy a forgalmi file-t végig kellene olvasni.

LISTA

A lista tulajdonképpen soros logikai elrendezést jelent (a rekordok elérhetősége szempontjából). Egy soros file-tól az különbözteti meg, hogy a rekordok fizikai elrendezése nem soros. Minden rekord tartalmazza a soron következő rekord mutatóját (címét, sorszámát, stb.). Tehát a rekordok bővülnek a következő rekord (listaelem) mutatójával. Ezen kívül tudnunk kell még a lista file-ról:

- melyik rekord a lista feje (első eleme)
- melyik rekord a lista utolsó eleme (vége)

Tehát:



Ahogy korábban emlitettük, minden szerkezetet relatív file-ként kezelünk. A lista szerkezetnél ez azt jelenti, hogy a listafej mutatója, ill. a következő elemek mutatói relatív rekordsorszámok. A lista végjeleként a legegyszerűbb zérót használni, mert ilyen relatív sorszám nincs.

A mutatókat célszerű kétbyte-os bináris számként tárolni. Egyrészt ez a legtömörebb tárolási forma (és a 65535 sorszám mindig elegendő), másrészt így a rekordműveleteknél sok esetben elkerülhetjük a bináris-decimális konverziót.

Pl:

IL$: a most beolvasott listaelem

```
RS$=RIGHT$(IL$,2) (a következő elem mutató)
PRINT#lf,"P"+CHR$(cs)+RS$+CHR$(1)
```

Tehát anélkül pozicionáltunk a lista következő elemére, hogy "tudnánk", mennyi a relatív rekordsorszáma a következő elemnek.

(Természetesen azt meg kell vizsgálnunk, hogy nem ért-e véget a lista. Ha RS$=CHR$(0)+CHR$(0), akkor végetért.)

Egy file - egy lista

A legegyszerűbb szerkezet, ha egy file-ban egy listát tárolunk. A gyakorlatban ez ritkán fordul elő, de nézzünk erre is egy példát:

Egy file feldolgozását két módon kell végeznünk:

- a rekordok file-ba kerülésének sorrendjében
- a rekordok egy mezőjének - mint kulcsnak - a sorrendjében, rendezve a kulcs szerint.

Tegyük fel, a file sok rekordból áll, a kulcsok hosszúak, tehát nem fér el az indexfile a memóriában.

A feldolgozásból adódóan (tegyük fel) floppyn sem tudunk külön kulcsfile-t használni.

Így az a lehetőség marad, hogy a keletkezés sorrendjében folyamatos relatív sorszámmal írjuk fel a rekordokat. A felíráskor a rekordhosszt 2 byte-al nagyobbra vesszük (ez lesz majd a következő elem mutatója a listában). Létrehozunk ideiglenesen egy kulcsfile-t (kulcs + rekord sorszám), majd ezt a kulcsra rendezzük.

A rendezett kulcsfile első rekordjában lévő relatív sorszámot külön tároljuk (ez lesz a lista fejének mutatója). A második rekordban lévő sorszámot beírjuk az első rekordban lévő sorszám szerinti adatrekordban fenntartott 2 byte-ba.

A harmadik rekordban levő sorszámot beírjuk a második rekordban lévő sorszám szerinti adatrekordba, stb. Így az adatfile egy olyan lista lett, ami biztosítja a szükséges feldolgozhatóságot, most már kulcsfile nélkül.

Ez a módszer csak akkor alkalmazható, ha az adatrekordokat az adatfile "listává alakítása" után már csak feldolgozni kell (pl. olvasni listázáshoz), karbantartani nem. Különben a hosszadalmas eljárást a mutatók létrehozására meg kellene ismételni.

(Természetesen ez a módszer a gyakorlatban ritkán jó. Az adatrekordban lévő kulcsmező és a következő elem mutató lényegében annyi helyet foglal el a diszketten, mint a külön kulcsfile. Egy előnye azért van. Az adatfile olvasása gyorsabb lesz, mint külön kulcsfile használatával, mert nem kell külön pozicionálnunk a kulcsrekord miatt. Ha a kulcsfile azon a drive-on van, mint az adatfile és a drive kis teljesítményű (pl. 1541-es), a feldolgozás még lassúbb, mert a drive-on egyszerre csak egy relatív file lehet nyitva. A kulcsrekord olvasása után a kulcsfile-t le kell zárnunk, az adatfile-t meg kell nyitni (kb. 1 sec), utána az adatfile-t lezárni, a kulcsfile-t megnyitni (újabb 1 sec), stb.)

Egy rekord több listában

Ha egy adatfile-t többféle képpen - több kulcs alapján - kell feldolgozni, létrehozhatunk külön kulcsfile-okat a feldolgozáshoz. A feldolgozások elvégezhetők a kulcsfile-ok szerint, de használhatjuk az előző módszert is. Most egy adatrekord több listának lesz eleme. A lista szerkezet kialakítását nem írjuk le. Csak annyiban különbözik az előzőtől, hogy kulcsonként kell elvégezni a mutatók beírását. Ez a szerkezet már nagyon merev, a file

bővítése, rekord törlése igen nehézkes. Ha a file-t a felépítés után karbantartani nem kell, csak feldolgozni a mutatókkal kialakított szekvencia szerint, a szerkezet egyszerű, gyors feldolgozást tesz lehetővé.

Példa:

Megint a készáru raktár kezelési feladat. Tegyük a forgalmi adatokat többféle rendezettség szerinti lista szerkezetbe egy karbantartási időszak végén. Az összesítő kimutatások - az egyes rendezettségeknek megfelelően - gyorsan elkészíthetők.

Az eljárás akkor igazán hatékony, ha az egyes rendezettségekhez több kimutatás készítése tartozik.

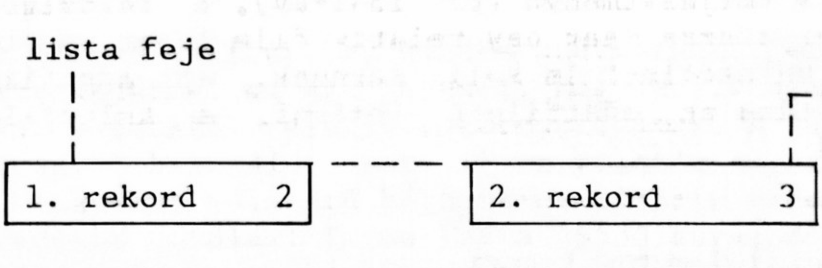
Így egy rendezés ideje több feldolgozásra oszlik el. A forgalmi file karbantartására a feldolgozási időszak lezárása után nincs szükség, tehát a file-szerkezet használatának ez a feltétele is teljesült.

Több lista egy file-ban

Elemezzük a következő feladatot. Egy elektronikus berendezéseket gyártó vállalatnál a gyártáshoz szükséges anyagokat törzsfile-ban tartják nyilván. A szokásos megjelölő és mennyiségi adatokon kívül szükséges az anyagot helyettesítő, hasonló anyagok nyilvántartása is (a hazai szállítási fegyelemből kiindulva ez tényleg szükséges).

A helyettesítő anyagok száma egy anyaghoz igen eltérő. Ha a törzsrekordban fix számú helyettesítő anyag azonosítójának megfelelő helyet tartanánk fenn, a törzsfile mérete jelentősen megnőne. A fenntartott hely nagyrésze kihasználatlanul maradna, esetleg egyes anyagoknál nem tudnánk az összes szükséges helyettesítő anyag azonosítóját tárolni. Kinálkozik az a megoldás, hogy azokhoz az anyagokhoz, melyeknek vannak helyettesítői, kapcsoljunk egy listát a helyettesítő anyag azonosítók tárolására. A listákat célszerű egy file-ban összefogni. A listák gyakran változnak, beszurunk, törlünk elemeket. A törölt rekordok "helyét" fel kell szabadítanunk beszurandó elemek számára (más listához tartozó elem számára is).

A háttér rugalmas kihasználását egy kezdeti listán keresztül valósíthatjuk meg. A listafile részére rendelkezésre álló területen létrehozunk egy olyan listát, amely az egész területet kitölti és a listaelemek mutatói sorban egymásra mutatnak. Ez a "szabad helyek" listája. A lista fejének mutatója induláskor az első rekord:



Ha a szabad helyek listájából elveszünk egy elemet, a lista feje most a következő elem lesz. Az elvett elemet építjük be a karbantartás alatt álló listafile-ba.

Ha a karbantartás alatt álló listafile-ból törlünk egy elemet, az lesz a szabad helyek listájának a feje, az ebben lévő mutató a szabadhely lista eddigi fejére mutat.

Az "igazi" listák fejének mutatója a törzsrekordban lehet (ha használunk indexfile-t, az indexrekordban is lehet).

Következzék a törzsrekordhoz tartozó lista karbantartásának algoritmusa:

0. Értékeld ki a következő tranzakció típusát
1. Ha bővíteni kell a listát, menj 100-ra
2. Ha törölni kell a listából, menj 200-ra
3. Ha módosítani kell egy lista elemét, menj 300-ra

100. Ha már van a listának eleme, menj 110-re
101. A szabad hely lista fejének mutatója alapján olvasd be az első szabad rekordot
102. A beolvasott rekord sorszámát írd be a törzsrekordba (ez lesz a lista feje)
103. A lista rekordba írd be a szükséges adatokat
104. A lista rekordban lévő következő elem mutatót írd be a szabad helyek lista fejének mutatójába
105. A lista rekord következő elem mutatójába írj zérót
106. Menj 0-ra
110. A lista elemeken végigmenve, keresd meg a lista utolsó elemét
111. A szabad hely lista fejének mutatóját írd be a megtalált elembe (ez lesz a következő elem mutatója)
112. Olvasd be a szabad rekordot a mutató alapján
113. Menj 103-ra

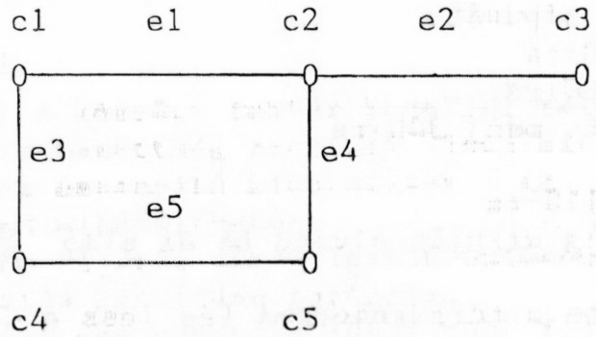
200. A lista elemeken végigmenve, keresd meg a törlendő elemet (közben mindig őrizd meg a rekord sorszámát, melyből a következő elem mutatót vetted)
201. Töröld a lista rekordot (pl. zéróval)
202. Ha a megtalált elem az első elem a listában, menj 210-re
203. Ha a megtalált elem az utolsó a listában, menj 220-ra
204. A megtalált elemben lévő következő elem mutatót írd be az előző lista elembe (ez a 200. pontban tárolva volt)
205. A megtalált elembe írd be a szabad hely lista fejének mutatóját
206. A szabad hely lista fejének mutatójába írd be a megtalált rekord sorszámát
207. Menj 0-ra
210. Írd a megtalált elem következő elem mutatóját az adatrekord listafej mutatójába
211. Menj 205-re
220. Az előző listaelem következő elem mutatójába írj zérót
221. Menj 205-re
300. A listaelemeken végigmenve, keresd meg a módosítandó elemet
301. Módosítsd a szükséges adatokat
302. Menj 0-ra

GRÁF, FA

Bevezetés

A struktúrák tárgyalása előtt ki kell térnünk néhány gráf-elméleti fogalomra.

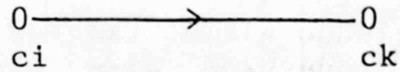
A gráf csúcsokból és élekből álló alakzat. Megadható grafikus formában is:



c1,c2,...,cn: a gráf csúcsai
e1,e2,...,en: a gráf élei

Azt az utat (egymásutáni éleket), melyen a gráf egyik csúcsából egy másikba eljuthatunk, vonalnak nevezzük.

Ha egy csúcsból kiindulva bejárhatunk egy olyan utat, hogy visszajussunk a kiinduló csúcsba, a vonalat körnek hívjuk. Ha a gráf csúcsaiból kiinduló éleket irányítással látjuk el, a gráf-ot irányított gráf-nak nevezzük:



Azt a csúcsot, melyből az él kiindul, kezdőcsúcsnak (kiinduló pontnak) nevezzük. Azt a csúcsot, melybe az él befut, végcsúcsnak (végpontnak) nevezzük.

Az olyan élt, melynek kiinduló és végpontja azonos, huroknak nevezzük.

Ha két csúcsot egynél több él köt össze, az éleket párhuzamos éleknek nevezzük.

Az olyan gráf-ot, melyben minden csúcspár között van él, teljes gráfnak nevezzük.

Ha a gráf-nak csak egy - az adott csúcsokkal és élekkel - meghatározott részét tekintjük, ezt részgráf-nak nevezzük. Az olyan gráf-ot,mely sem párhuzamos, sem hurok élt nem tartalmaz, egyszerű gráf-nak nevezzük.

A fa olyan (irányított, vagy irányítatlan) egyszerű, összefüggő gráf, mely nem tartalmaz kört. A fának, mint gráf-nak az éleit ágaknak nevezzük. Ha a fa irányított gráf, azt a csúcsot, melyből az ágak kiindulnak, gyökérelemnek nevezzük.

A gráf, mint file-struktúra

A struktúrát egy példán keresztül elemezzük:

Egy gépgyár termékeibe sok olyan alkatrészt épít be, melyek több termékben is benne vannak. A gyár termékeihez rendelésre alkatrészeket is gyárt.

Egységes azonosító rendszert vezettek be az összes termék és alkatrész azonosítására.

Megoldandó a termékek szerelési utasításának gépi nyilvántartása. A szerelési utasításnak tartalmaznia kell, hogy az egyes alkatrészekből menynyit és milyen sorrendben kell felhasználni a szereléshez. Követelmény to-

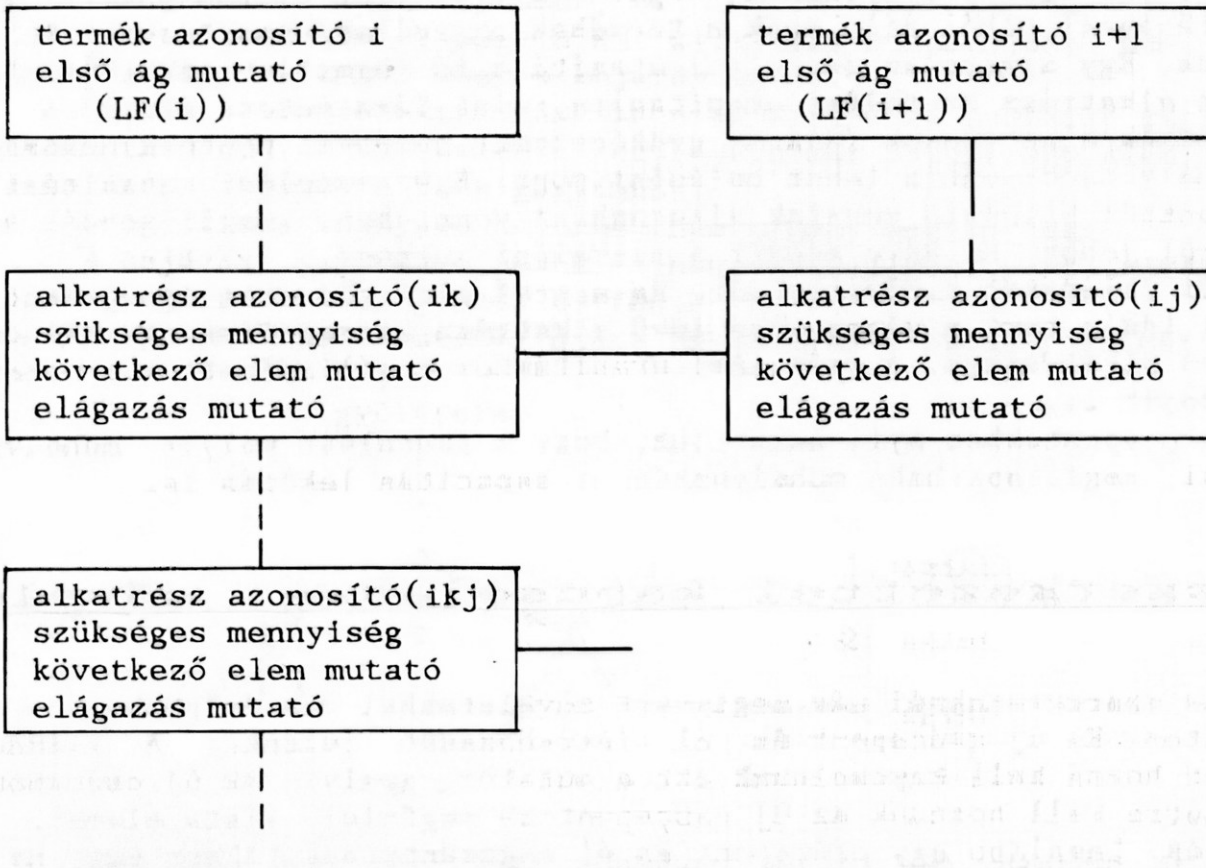
vábbá, hogy a számítógépes rendszer egyedi alkatrész rendelés esetén is el tudja készíteni a szerelési utasítást.

Az alkatrész megrendelésénél ismert, hogy az alkatrész melyik termékhez tartozik.

A szerelési utasításokat graf-ként tároljuk. Mindegyik termékhez tartozó szerelési utasítás egy részgráf (fa) lesz. A gráf tárolásához lista file-szerkezetet használunk, a korábban említett "egy file-ban több lista" formát használjuk fel. A lista elem felépítése:

- az alkatrész egyedi azonosítója
- szükséges mennyiség az alkatrészből a következő nagyobb részegység szereléséhez
- elágazás mutató (ha az alkatrészt több más alkatrészből szerelik össze)
- következő elem mutató.

A termék azonosítókat külön állományban (indexfile-ban) tároljuk, azzal a relatív rekordsorszámmal együtt, amelyik az első ágra mutat.



A felépítés kapcsolata a gráf-al:

- A fák gyökérelemei a termék azonosítói
- A fák irányítottak. Az élek adják meg, hogy az alkatrészből hány kell a termékhez.
- A szerelési sorrendet a kezdőpontokból kiinduló élek végpontjainak elhelyezkedése adja. A csúcspontból kiinduló élek végpontjait listában fogjuk össze.
- Minden lista elem egyben kezdőpont is lehet, kiindulhatnak belőle élek.

Térjünk vissza a listaelem felépítésére. Az alkatrész azonosítója kapcsolatot teremthet egy másik file-val. Pl. a félkészáru raktárban az alkatrészeket szintén így azonosítjuk.

Egy másik file-ban van a félkészáru raktár nyilvántartása. A szerelési utasítás készítésekor ezt a file-t is felhasználva pl. kiszámíthatjuk a

termék árát, vagy ellenőrízhetjük, rendelkezésre áll-e elegendő alkatrész a gyártáshoz.

Az elágazás mutató az alkatrészből - mind kezdőpontból - kiinduló lista fejének mutatója.

A - már ismert - következő elem mutató a lista következő elemére mutat, vagyis a következő alkatrészre a szerelési sorrendben.

Szükséges mennyiség. Itt lesz az eltérés a gráf-tól. A gyökér elemtől a legalsó ágig vezető vonalon nem az élek hosszának összege adja a szükséges mennyiséget a termékhez, hanem az élek szorzata.

A termék szerelési utasításának elkészítéséhez tehát egy indexfile-ból ki kell keresnünk a termék azonositóját. E mellett találunk egy mutatót,ami egy fa gyökérelemére mutat. A szerelési utasítás elkészítése nem más, mint a fa "bejárása" adott algoritmus szerint.

Egy alkatrész szerelési utasitásának elkészitéséhez ismerjük a terméket, amihez az alkatrész tartozik. Ez sem jelent mást, a fa bejárását. A listázást csak akkor kell elkezdenünk, ha már eljutottunk a keresett alkatrészhez.

Tehát összefoglalva, a feladat megoldására gráf-ot használunk. A gráf független részgráf-okból áll, ezek a termékek szerelési utasításai. A fák irányítottak. Egy alkatrész szerelési utasítása is független részgráf (eltekintve az alkatrész szerelési utasítására, mint fára mutató éltől).

A termékek/alkatrészek fájának gyökérelemét belépési pontnak nevezzük. A gráf bármely csúcspontja lehet belépési pont. Egy szerelési utasítást a belépési pontból kiinduló vonalak alkotnak. A vonalakon - megállapodás szerint - balról jobbra haladva adódik a szerelési sorrend.

A fenti feladatot kibővíthetjük. Ha a gráf élei pl. azt is jelentik, hogy mennyi ideig tart a végpontban lévő alkatrész rászerelése a kiinduló pontban lévő alkatrészre, a szerelési utasításhoz megállapítottuk a szerelés normaidejét is.

Ha a csúcspontokban nyilvántartjuk, hogy a szerelést melyik mühelyben kell végezni, megállapítható mühelyenként a kapacitás lekötés is.

A fa szerkezettel kapcsolatos műveletek

Itt is a más szerkezeteknél már megismert műveletekkel van dolgunk:

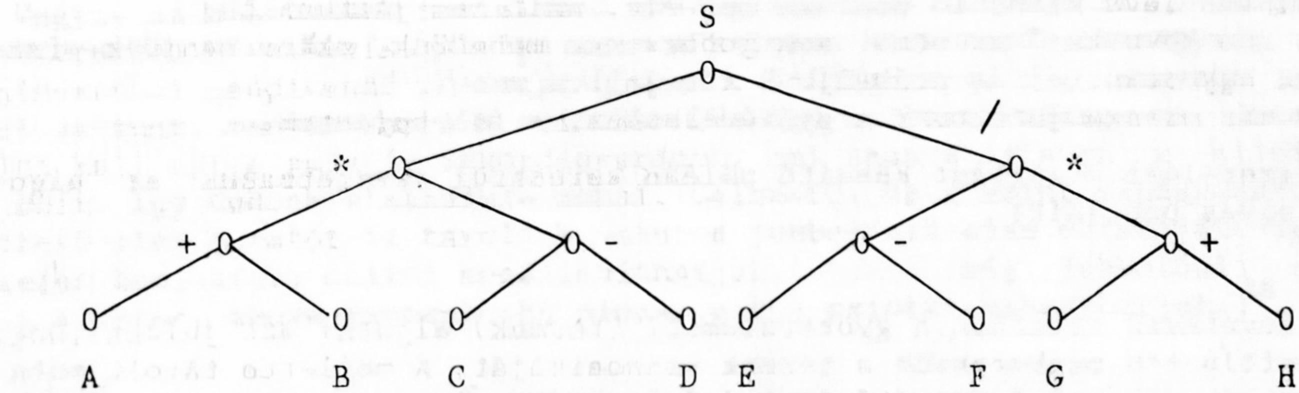
- Bővítés. Ez új csúcspont és él létrehozását jelenti. A kiinduló csúcsponthoz hozzá kell kapcsolnunk azt a mutatót, amelyik az új csúcspontra mutat. Létre kell hoznunk az új csúcspontnak megfelelő lista elemét.

- Törlés. Legalább egy csúcspont és él megszüntetése. Akkor egy, ha a megszüntetendő csúcspont nem élek kezdőpontja is egyben. Törölnünk kell azt a mutatót a kiinduló csúcspontnál, amelyik a törlendő csúcspontra mutat. A törlendő elemet fel kell szabadítanunk (hozzá kell fűzni a szabad helyek listájához). Ha a törlendő csúcspont élek kezdőpontja is egyben, a csúcspontból kiinduló részfát kell törölnünk.

- Rekord (csúcspont) keresése. Egy csúcspontot általában nem tudunk közvetlenül elérni. Visszatérve az előző fejezetben leírt feladathoz, csak a termékek mutatóját tároltuk külön indexfile-ban. Ha egy alkatrészt keresünk, a terméktől - mint belépési ponttól - be kell járnunk a fát addig, amíg el nem érünk a keresett alkatrészhez. Minden műveletnél szükséges a fa bejárása. A bejárás azt jelenti, hogy végigmegyünk a fa vonalain úgy, hogy egy csúcspontot csak egyszer veszünk figyelembe. A bejárás módját (algoritmusát) definiálni kell, a fa, mint adatszerkezet csak így definiált egyértelműen.

Nézzünk erre egy egyszerű példát:

Ábrázoljuk az S=(A+B)*(C-D)/(E-F)*(G+H) kifejezést "bináris" fával:

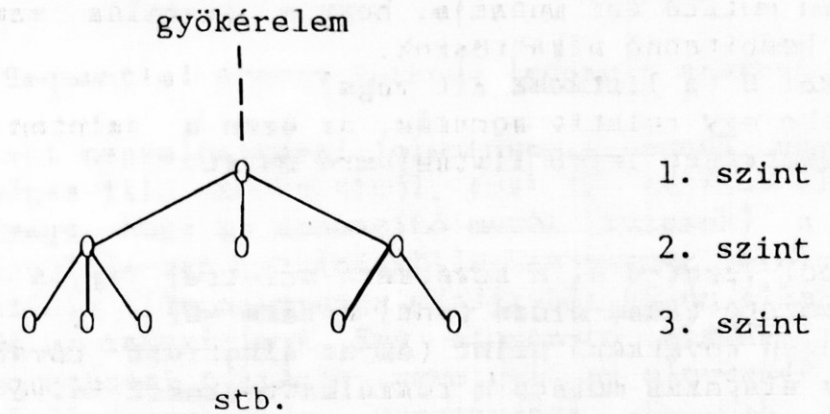


Ha ezt a fát balról-jobbra járjuk be, akkor jutunk a fenti kifejezéshez. Ha a bejárást jobbról-balra végezzük, ezt a kifejezést kapjuk: S=(H+G)*(F-E)/(D-C)*(B+A). Nyilvánvaló, hogy a két kifejezés kiszámítása nem ad azonos eredményt. Vagyis nem mindegy a bejárás módja.

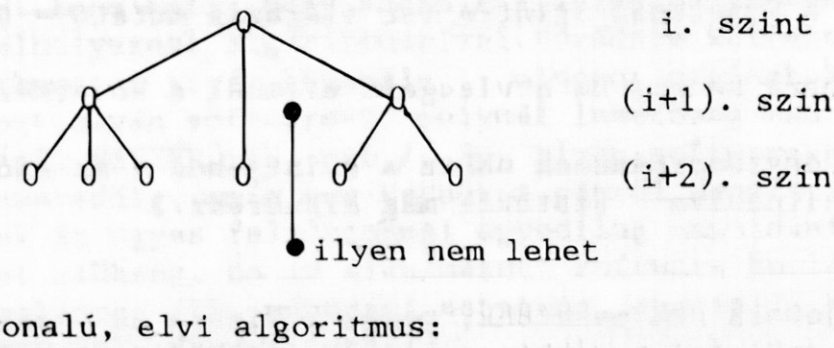
A termék szerelési utasításának elkészítéséhez használt gráf-ot lista-file-ban tároltuk. A listaelemek kezelésére már adtunk egy algoritmust. Most a fa bejárására adunk elvi algoritmust.

Az algoritmus könnyebb megértéséhez néhány megjegyzés:

- a bejárást a gyökérelemből kiindulva (belépési pont) végezzük
- a bejárás balról jobbra történik
- a könnyebb kezelhetőség miatt bevezetjük a szint fogalmat:



- kikötjük, hogy egy vonal csúcspont nélkül nem vezethet át szinten



A nagyvonalú, elvi algoritmus:

a/ induljunk el a gyökérelemtől a következő szintre vezető élek közül a baloldalin

b/ ha az él végpontjából vezet él a következő szintre, akkor induljunk el ezek közül a baloldalin

c/ ha a végpontból nem vezet el a következő szintre, akkor próbáljunk jobbra menni ezen a szinten (ez akkor lehetséges, ha van az előző szinten lévő kiinduló pontnak még éle, amit nem jártunk be)

d/ ha sem következő szintre, sem jobbra nem mehetünk, akkor menjünk vissza egy szinttel és próbáljunk itt jobbra menni

e/ ha már vissza jutottunk a gyökérelemhez, a fát bejártuk.

A szerelési utasítást készítő példán keresztül részletezzük az algoritmus egyes pontjait:

ad a/

A következő szinthez a gyökérelemtől (termék) eljutni azt jelenti,hogy az indexfile-ban megkeressük a termék azonosítóját. A mellette tárolt mutató (a termék szerelési utasítását leíró fa gyökérelemére mutat egy lista szerkezetű file-ban) alapján beolvassuk a termék főegységei közül az első beépítendő alkatrészt leíró elemet.

Tehát a főegységeket egy lista tartalmazza. A listában az alkatrészeket a beépítés sorrendjében vettük fel. A listaelem tartalma pl.:

1. az alkatrészek jellemzőit leíró file-ban ennek az alkatrész rekordnak a relatív sorszáma
2. az alkatrész azonosítója
3. a termék azonosítója (az indexfile-ban is ez van), amihez ez az alkatrész tartozik
4. elágazás mutató (azt mutatja, hogy ez az alkatrész áll-e még részalkatrészekből, ha nem, értéke: 0
   ha igen, értéke egy relatív sorszám, a részalkatrészeket leíró lista első elemére (a lista fejére) mutat
5. következő elem mutató azt mutatja, hogy a szerelés ezen szintjén vannak-e még beépítendő alkatrészek,
   ha nem, értéke: 0 (a listának itt vége)
   ha igen, értéke egy relatív sorszám, az ezen a szinten következő, beépítendő alkatrészt leíró listaelemre mutat.

ad b/

Az él végpontjából vezet-e él a következő szintre? Vagyis a vizsgált elemben az elágazás mutató (lásd előző pont) értéke =0?

Ha van ezen az ágon következő szint (ez az alkatrész további részalkatrészekből áll), az elágazás mutató a részalkatrészeket leíró lista fejére mutat (a következő szint baloldali éle).

ad c/

Ha nem vezet el a következő szintre (az elágazás mutató = 0), menjünk jobbra, ha tudunk.

Akkor tudunk jobbra menni, ha a vizsgált elemnél a következő elem mutató <> 0.

(Az alkatrészek összeépítésének ehhez a szintjéhez - az előző, felsőbb szintű alkatrészből kiindulva - tartozik még alkatrész.)

ad d/

Ha sem le, sem jobbra nem mehetünk, menjünk vissza és jobbra. Vagyis az alkatrész már nem épül fel további részalkatrészekből és az őt tartalmazó részegység sem áll további alkatrészekből.

(A vizsgált listaelemnél az elágazás mutató és a következő elem mutató is zéró.)

Menjünk egy szintet vissza és itt próbáljunk jobbra menni.

Vagyis az alkatrészt tartalmazó részegység után következő részegység összeszerelése következik. Ha van azon a szinten következő részegység. Ha már nincs több beépítendő részegység ezen a szinten sem, még nagyobb egységnél tartunk, menjünk erre a szintre. (Ha egy szintet elhagyunk, mindig tárolni kell azt a relatív rekordsorszámot, ami azon a szinten a kiinduló elem volt. Így tudunk visszafelé menni. Célszerű, ha a szint elhagyásakor a következő elem mutatót is tároljuk, mint a jobboldali elem mutatóját. Így a listaelem beolvasása nélkül megállapíthatjuk, van-e még jobboldali elem azon a szinten, ahová mennénk. Ha nincs, ezt a szintet kihagyhatjuk.)

ad e/

Visszajutottunk a gyökérelemhez. Vagyis a szerelési utasítás elkészült.

Az algoritmus program szintű kidolgozását az olvasóra bizzuk.

Az algoritmus segítségével bejárhatjuk a fa egy részét is, a csúcspontok közül bármelyik lehet egy részfának gyökere. (Az algoritmust addig végezzük, amíg a listaelemben megtaláljuk a gyártandó alkatrész azonosítóját. Ahol megtaláltuk, az a listaelem lesz a gyökérelem. Tehát a szükséges tevékenységeket /ami miatt a bejárást végezzük/ innen kell indítani. Az algoritmus akkor ér véget, ha újra visszajutottunk ehhez az elemhez.)

ISAM

ISAM - Indexed Sequential Access Method, indexelt szekvenciális elérési mód

Az ISAM gyakorlati megvalósításai jelentősen eltérnek egymástól, nem beszélhetünk egységes file szerkezetről, csak egy sajátos elérési módról. Az elérési mód lényege, hogy az azonosító mezők (kulcsok) a rekordokból kiemelve, egy külön file-ban, az indexfile-ban vannak tárolva. Tehát az indexelt szekvenciális file szervezés mindig két külön állományt tételez fel. Az adatfile-t és az indexfile-t. Egy azonosító elérése az indexfile-ban szekvenciális kereséssel történik, erre utal az elnevezés is.

Fentiek a file szervezésére vonatkoznak. Nagyobb számítógépeken az alapsoftware része az indexelt szekvenciális file kezelés. A felhasználónak a magas szintű nyelvekben olyan utasítások állnak rendelkezésére, melyekkel a file-t anélkül kezelheti, hogy annak fizikai elhelyezésével, a rekordok és az indexek elhelyezési algoritmusaival törődnie kellene. A C 64-nél - ha ilyen file szerkezetet kíván használni - mindent magának kell megírnia.(Illetve beszerezhet olyan software-t, melynél lehetőség van a logikai szintű adatkezelésre /pl. MASTER, IS, stb./. Egy ilyen software eleinte igen nagy segítség. Egészen addig, amíg nem tudunk a gépről annyit, hogy magunk is megpróbálkozzunk az egyes feladatoknál egyedileg megoldani a file-kezelést. Erre akkor lehet szükség, ha az alkalmazott software korlátai a feladat miatt elfogadhatatlanok. /Pl. működési sebesség, speciális karbantartás, nagy rekord szám, nagy rekordhossz, stb./)

Foglaljuk össze, hogy mi a minimális követelmény az indexszekvenciális állományok kezelésével kapcsolatban:

- bármely rekord közvetlen elérése az indexe (kulcsa) alapján
- a file olvasása szekvenciálisan az indexek rendezett sorrendjében (esetleg bármely indextől kezdve is)
- a file bővíthetősége (rekord beépítése)
- törlés a file-ból (rekord megszüntetése)
- módosítás (rekord felülírása más tartalommal).

Ha a memória file-okról szóló fejezetet mégegyszer átolvassuk, látjuk, hogy a legegyszerűbb esetben az indexfile a memóriában lehet.

A hivatkozott fejezetben csak az indexfile rendezettségére (rendezésére) nem tértünk ki a szükséges feladatok közül. Ha ezt is biztositjuk, ez már indexelt szekvenciális file szerkezet. A rendezettséget biztosíthatjuk úgy, hogy új index belépése esetén, vagy törléskor a belépési (törlési) helytől a file-t "eltoljuk" a memóriában.

Másik lehetőség, hogy a rendezettséggel nem törődünk a karbantartáskor. A rendezést a karbantartás végén, az indexfile floppyra visszaírása előtt végezzük el. Az indexfile-nak a memóriában tárolása a legegyszerűbb és a leggyorsabb feldolgozást teszi lehetővé.

Ha a memóriában nincs elegendő hely az indexfile számára, floppyn kell tárolnunk (ez már egy "igazi" indexelt szekvenciális szerkezet.) Az indexek hossza általában 1-2 nagyságrenddel kisebb, mint az adatrekord mérete. A keresés sokkal gyorsabb, mint szekvenciális adatfile esetén. Nagyobb állományok esetén az indexfile megnövekedett mérete miatt a szekvenciális keresés már nem gazdaságos, ilyenkor az indexfile-t alkalmasan megválasztott méretű csoportokra osztva, egy magasabb szintű indexfile épül fel, amely az alacsonyabb szint egy-egy azonosító csoportját indexeli.

Belátható, hogy a minimális elérési idő (olvasáskor) úgy érhető el, ha egy-egy indexszinten belül a szekvenciális keresés nem lépi át a szektorhatárt, tehát minden indexszintből egy és csak egy szektort olvasunk be.

Egy rekord eléréséhez általában nem kell sok olvasás, amit a következő példa szemléltet:

Tegyük fel, hogy létrehoztunk egy indexelt file-t, amely 15000 rekordot tartalmaz. Az azonosító (kulcs) hossza 8 byte, a pointerrel együtt tehát 10 byte. Egy indexszektoron 25 azonosító fér el.

Egy alacsonyabb indexszint ebből következően 25-ször több azonosítót tartalmaz, mint a közvetlenül felette álló indexszint. A legmagasabb szint 25 azonosítót (indexet) tartalmaz, a következő 25*25-öt, az ez alatt lévő 25*25*25-öt. Ez már 15625, tehát több, mint a rekordszám. Ha a főindexszektort a RAM-ban tároljuk, 2 index + 1 adatszektor olvasással elérhetjük a 15000 rekord közül bármelyiket.

Egy új index beszurásakor - az állomány bővítésekor - már nem olyan kedvező a helyzet, mint az olvasáskor (adat elérésekor). Legrosszabb esetben az történhet, hogy minden indexszintet egy új indexszektorral kell bővíteni, mert az a szektor, ahova az új azonosítót be kellene szurni, már tele van. Az új szektorhoz a magasabb szinten is be kell szurni egy új indexet, amely őrá mutat, és így tovább. A fenti példában, a legkedvezőtlenebb esetben egy új index beszurása 3 olvasást, 2 új szektor allokálást és 5 írást jelent, ami adatrögzítés közben már nehezen elviselhető várakozási idővel jár.

Ennek ellenére - ha a feladat az adatrekordok közvetlen elérését követeli meg - az azonosító file pedig nem fér be a RAM-ba , az indexszekvenciális file szerkezet elég hatékony eszköz. Pl. egy könyvtárban a könyvek nyilvántartásához a viszonylag lassú adatfelvitel megengedhető, ha a keresett könyvnek megfelelő rekordot gyorsan elérhetjük.

Az indexszekvenciális file előnye még, hogy az állomány szekvenciális elérését is biztosítja, ha a legmélyebb indexszint szektorait láncoljuk (listába fogjuk össze). Így pl. a hivatkozott könyvtári nyilvántartás alapján gyorsan készíthetünk katalógust a könyvek azonosítója szerint.

PÉLDA

A file-kezelés bemutatására készítettünk egy egyszerű példát.

A példa alap BASIC-ben van írva, bizonyítván, hogy egy feladat így is megoldható, ha kis állományokról van szó és nem vagyunk nagyon igényesek a sebesség szempontjából. A példa megmutatja, milyen rutinokat kell készítenünk. (A rutinok más feladatnál is jók lesznek.) Érdemes egy saját feladat megoldása előtt tanulmányozni ezt a mintafeladatot és meggondolni, mely rutinokat lenne érdemes megírni assemblerben.

A példa egy személyi nyilvántartás, pl. egy orvos nyilvántartása magánbetegeiről. A pácienseket személyi számukkal azonosítjuk, és az alábbi adataikat tároljuk:

- személyi szám (11 karakter)
- név (20)
- lakcím (40)
- telefon (6)
- kórisme (40)

Az azonosítókat (a személyi számokat) a lemezen egy indexfile-ban, a program futása közben pedig a memóriában, egy BASIC tömbben tároljuk.Az indexfile rendezett.

Az adatlemezt előkészítő program:

```
20000 UN=8:          REM KÉSZÜLÉK
20002 RH=122:        REM REKORDHOSSZ
20004 MP=400:        REM MAX. REKORDSZÁM
20010 OPEN 115.UN,15,"I"
20015 GOSUB 1000:REM HIBACSATORNA OLVASÁSA
20020 REM
20030 REM LEMEZ AZONOSÍTÓ ELLENŐRZÉSE
20040 REM
20050 OPEN 5,UN,2,"#":      REM DIREKT FILE
20060 PRINT#115,"U1:2,0,18,0":GOSUB 1000:REM BAM OLV.
20070 PRINT#115,"B-P:2,144":REM LEMEZAZONOSÍTÓ A BAM-BAN
20080 VO$=""
20090 FOR I2=1 TO 12
20100 : GET#5,A$:VO$=VO$+A$:NEXT
20110 IF VO$="PATIENT.ADAT" THEN 20140
20120 ?"NEM A JO LEMEZ VAN BENN"
20130 CLOSE 115:WAIT 198,15:GET A$:RUN
20140 CLOSE 5
20150 REM
20160 REM RELATÍV FILE FELÁLLÍTÁSA
20170 REM
20180 OPEN 4,UN,3,"0:PATIENT,L,"+CHR$(RH):GOSUB 1000
20190 PN=MP:GOSUB 2000:           REM REL. REKORD POZICIONÁLÁS
20200 PRINT#4,CHR$(255);:CLOSE 4
20210 OPEN 4,UN,3,"0:PATIENT":GOSUB 1000
```

```
20220 PN=MP:GOSUB 2000:GOSUB 1000:REM ELLENŐRZÉS
20230 CLOSE 4
20240 REM
20250 REM INDEXFILE FELÁLLÍTÁSA
20260 REM
20270 OPEN 3,UN,3,"0:PATIENT.INDEX,S,W":GOSUB 1000
20275 PRINT#3,1:REM Az EDDIG HASZNÁLT LEGNAGYOBB REK. SORSZÁM
20280 PRINT#3,"ππππππππππ,1":REM CHR$(" ")=255
20290 CLOSE 3:GOSUB 1000
20300 CLOSE 115:CLR:PRINT "*** OK":END
```

A hibacsatorna olvasó rutin:

```
1000 INPUT#115,CD,MG$,TK$,SC$:IF CD<20 THEN RETURN
1010 PRINT CD;MG$;" ";TK$;" ";SC$
1020 STOP:RETURN
```

A relatív rekord pozícionáló rutin: (a relatív rekord sorszám a PN változó-ban)

```
2000 PRINT#15,"P";CHR$(FN L(PN));CHR$(FN H(PN));CHR$(1)
2010 RETURN
```

Az FN L és az FN H függvényeket a program elején definiáljuk:

```
500 DEF FN H(J)=J/256
510 DEF FN L(J)=J-256*INT(J/256)
999 GOTO 20000
```

A felvivő-módosító-lekérdező program (1000-2010-ig azonos az előző programmal)

Az indexállományt kezelő rutinok:

```
3000 REM
3010 REM INDEXFILE BEOLVASÁSA BASIC-TÖMBBE
3020 REM
3030 MP=400:DIM AZ$(MP),PN%(MP)
3040 REM    AZ AZ$ TÖMB A SZEMÉLYI SZÁMOKAT, A PN% TÖMB A
3050 REM    RELATÍV REKORDSORSZÁMOKAT TARTALMAZZA
3060 OPEN 3,UN,3,"0:PATIENT.INDEX,S,R":GOSUB 1000
3070 INPUT#3,DB:REM AZ EDDIG HASZNÁLT LEGNAGYOBB REK. SORSZÁM
3080 FOR I0=0 TO MP
3090 : INPUT#3,AZ$(I0),PN%(I0)
3100 : IF ST=0 THEN PRINT I0;:POKE 211,0:NEXT
3110 IF ST<>64 THEN ? "INDEXFILE HIBA":STOP
3120 PRINT I0:MI=I0
3130 CLOSE 3:RETURN

4000 REM
4111 REM INDEXFILE KIÍRÁSA
```

```
4020 REM
4030 OPEN 3,UN,3,"0:INDEX.TEMP,S,W":GOSUB 1000
4040 PRINT#3,DB
4050 FOR I0=0 TO MI
4060 : PRINT#3,AZ$(I0);",";MID$(STR$(PN%(I0)),2,7)
4065 : PRINT I0;:POKE 211,0:NEXT
4070 GOSUB 1000
4080 CLOSE 3:GOSUB 1000
4090 PRINT#115,"S0:PATIENT.INDEX":GOSUB 1000
4100 PRINT#115,"R0:PATIENT.INDEX=INDEX.TEMP":GOSUB 1000
4110 RETURN
```

A következő rutin az SZ$ változóban adott személyi számot keresi az AZ$ tömbben. Ha megtalálta, OK értéke -1 lesz, ha nem, 0. Az AP változóba vagy a megtalált személyi szám tömbindexe kerül, vagy az az index, amely a keresett személyi szám indexe lenne a (rendezett) tömbben. A keresés algoritmusa az ún. bináris keresés. A bináris keresés elve: a AP,ZZ-re teljesül, hogy

AZ$(AP) <= SZ$ <= AZ$(ZZ),

Felezzük meg az (AP,ZZ) intervallumot. A két fél közül az egyikre ismét teljesül a fenti összefüggés. Azt felezzük meg újra és így tovább.

```
5000 AP=0:ZZ=MI
5010 GOSUB 5500
5020 IF SZ$>AZ$(AP) THEN AP=AP+1
5030 RETURN
5500 FOR I0=0 TO 10
5510 : IF AP>=ZZ THEN OK=(SZ$=AZ$(AP)):RETURN
5520 : J=INT((AP+ZZ)/2)
5530 : IF SZ$>AZ$(J) THEN AP=J+1:NEXT
5540 : IF SZ$<AZ$(J) THEN ZZ=J-1:NEXT
5550 AP=J:OK=-1:RETURN
```

Az I0 ciklus soha nem fut ki 10-ig, s I0 értékét nem is használjuk. A ciklus csak azért kell, mert a FOR-NEXT lényegesen gyorsabb, mint a GOTO.

A 6000, 7000 rutinok egy személyi számot törölnek, ill. szúrnak be az indextömbbe. A törölt személyi számhoz tartozó relatív rekord újra fel lesz használva egy új belépéskor, mégpedig úgy, hogy törléskor a személyi szám helyére CHR$(255)-ket, " "-ket írunk és áttesszük a tömb végére:

```
6000 REM
6010 REM AZONOSÍTÓ TÖRLÉSE AZ INDEXTÖMBBŐL
6020 REM
6100 PN=PN%(AP)
6115 IF AP=MI THEN 6130
6110 FOR I0=AP TO MI-1
6120 : AZ$(I0)=AZ$(I0+1):PN%(I0)=PN%(I0+1):NEXT
6130 AZ$(MI)="ππππππππππ":PN%(MI)=PN
6140 X=FRE(0):RETURN
```

```
7000 REM
7010 REM BESZURÁS AZ INDEXTÖMBBE
7020 REM
7030 IF MI<MP OR AZ$(MI)>"π" THEN 7120
7040 PRINT CHR$(19)"NEM FÉR EL TÖBB";:REM CHR$(19)=HOME
7050 WAIT 198,15:GET A$
7060 PRINT CHR$(19)"
7070 OK=0:RETURN
7120 IF AZ$(MI)>"π" THEN PN=PN%(MI):MI=MI-1:GOTO 7140
7130 DB=DB+1:PN=DB
7140 IF MI<AP THEN 7160
7145 FOR I0=MI TO AP STEP -1
7150 : AZ$(I0+1)=AZ$(I0):PN%(I0+1)=PN%(I0):NEXT :REM HELYET
7151 :                                          REM CSINÁL
7152 :                                          REM UJ ELEM-
7160 AZ$(AP)=SZ$:PN%(AP)=PN:MI=MI+1            :REM NEK ÉS
7170 OK=-1:X=FRE(0):RETURN:                     REM BEIRJA
```

Az X=FRE(0) utasitás azért van, hogy a "szemétgyűjtő" eljárás itt, ezeknek az amúgy is hosszabb futási idejű rutinoknak a végén működjön, s ne pl. egy mező rögzítése közben, megzavarva a program kezelőjét.

Ha az állományba sok személyt vettünk már fel, a beszúrás elég hosszadalmas. (Kb. 10 mp.) Megfigyelhetjük azonban, hogy a program jóval rövidebb idő alatt veszi nyilvántartásba a fiatal nőket, mint az idős férfiakat. A jelenség magyarázatát az Olvasóra bízzuk.

```
20000 REM
20010 REM FŐPROGRAM
20020 REM
20030 PRINT CHR$(147);:REM KÉPERNYŐ TÖRLÉS
20040 PRINT LEFT$(KL$,8);
20050 PRINT TAB(8);"1 ...FELVITEL"
20060 PRINT TAB(8);"2 ...MODÓSÍTÁS"
20070 PRINT TAB(8);"3 ...TÖRLÉS"
20080 PRINT TAB(8);"4 ...LEKÉRDEZÉS"
20090 PRINT TAB(8);"5 ...VÉGE"
20100 PRINT LEFT$(KL$,14); :REM A KL$ EGY HOME ÉS 24 KURZOR
20101 :                     REM LE KARAKTERT TARTALMAZ
20110 INPUT "FUNKCIÓ";FU
20120 IF FU<1 OR FU>5 OR FU<>INT<FU> THEN 20100
20130 ON FU GOSUB 10000,11000,12000,13000,14000
20140 GOTO 20000
```

A 10000-13000 rutinokról: Minden, a képernyőn rögzítendő adatmezőhöz tartozik egy DATA utasítás, pl.:

```
50000 DATA 4,1,"SZEMELYI SZAM",16,11
```

A 4. sor 1. pozícióba kerül a "SZEMELYI SZAM" felírat; a 4. sor 16. pozíciójától 11 hosszan kérjük be a személyi számot. (Vo. 31000 szubrutin)

```
10000 REM
10010 REM FELVITEL
10020 REM
10030 PRINT CHR$(147);:REM KÉPERNYŐ TÖRLÉS
10100 RESTORE
10110 READ LI,01,TX$,02,MH:REM MEZŐLEÍRÓ:SZEMÉLYI SZÁM
10115 IF FU<>1 THEN 10190 :REM FUNKCIÓ NEM FELVITEL,
10116 :                    REM SZEM. SZ. NEM KELL
10120 GOSUB 31000          :REM KÉPERNYŐ INPUT
10130 SZ$=DF$:IF SZ$="" THEN RETURN
10140 GOSUB 5000:IF NOT OK THEN 10150
10142 PRINT CHR$(19)"FEL VAN MAR VÉVE";:
10144 WRIT 198,15:GET A$
10146 PRINT CHR$(19)"                    ":GOTO 10120

10150 IF LEN(SZ$)<>MH THEN 10120:REM MIND A 11 JEGYET MEG
10151 :                          REM KELL ADNI
10160 IF LEFT$(SZ$,1)<>"1" AND LEFT$(SZ$,1)<>"2" THEN 10120
10170 GOSUB 30000                :REM NUMERIKUS MEZŐ?
10180 IF NOT NM THEN 10120

10190 READ LI,01.TX$,02,MH       :REM NÉV
10200 GOSUB 31000
10210 NE$=DF$:IF NE$="" THEN 10200

10220 READ LI,01,TX$,02,MH       :REM CÍM
10230 GOSUB 31000
10240 CI$=DF$

10250 READ LI,01,TX$,02,MH       :REM TELEFON
10260 GOSUB 31000
10270 TE$=DF$:IF TE$="" THEN 10300
10280 IF LEN(TE$)<>6 THEN 10260
10290 GOSUB 30000:IF NOT NM THEN 10260

10300 READ LI,01,TX$,02,MH       :REM KÓRISME
10310 GOSUB 31000
10320 DG$=DF$

10330 READ LI.O1,TX$,O2,MH       :REM OK (I/N) KÉRDÉS
10340 GOSUB 31000
10350 IF DF$="N" THEN 10100
10360 IF DF$<>"I" THEN 10340

10370 :                          REM HA A FUNKCIÓ FELVITEL,
10370 :                          REM BESZURÁS AZ INDEXTÖMBBE
10375 IF FU=1 THEN GOSUB 7000:IF NOT OK THEN RETURN
```

Minden megvan: bekértük a mezőket, elhelyeztük az azonosítót az indextömbben, a 7000-es szubrutin a relatív rekordsorszámot is megadta, már csak ki kell írni a rekordot:

```
10380 PN=PN%(AP):GOSUB 2000:GOSUB 1000:REM REL. REK. POZ.
10390 PRINT#4,NE$;",";CI$;",";TE$;",";CHR$(13);DG$
10400 GOSUB 1000
10410 IF FU=1 THEN 10000        :REM HA FELVITEL, VISSZA AZ
10420 RETURN                    :REM ELEJÉRE, HA NEM, RETURN
```

A módosítás rutin mindössze 4 sor:

```
11000 REM
11010 REM MÓDOSÍTÁS
11020 REM
11030 GOSUB 13000               :REM LEKÉRDEZÉS
11040 IF SZ$="" THEN RETURN
11070 GOSUB 10100               :REM MÓDOSÍTÁS A NÉVTŐL KEZDVE
11080 GOTO 11000
```

A törlés is egyszerű:

```
12000 REM
12010 REM TÖRLÉS
12020 REM
12030 GOSUB 13000               :REM LEKÉRDEZÉS
12040 IF SZ$="" THEN RETURN
12050 READ LI,01,TX$,02,MH      :REM OK (I/N)?
12060 GOSUB 31000
12070 IF DF$="N" THEN RETURN
12080 IF DF$<>"I" THEN 12060
12090 GOTO 6000                 :REM AZONOSÍTÓ TÖRLÉS ÉS RETURN
```

```
13000 REM
13010 REM LEKÉRDEZÉS
13020 REM
13025 PRINT CHR$(147);          :REM KÉPERNYŐ TÖRLÉS
13030 RESTORE
13040 READ LI.O1,TX$,O2,MH      :REM SZEMÉLYI SZÁM
13050 GOSUB 31000               :REM BEKÉRÉS
13060 SZ$=DF$:IF SZ$="" THEN RETURN
13070 GOSUB 5000:IF OK THEN 13200:REM HA VAN ILYEN
13080 PRINT CHR$(19)"NINCS ILYEN";
13090 WAIT 198,15:GET A$
13100 PRINT CHR$(19)"           ":GOTO 13051
```

Ha a keresett személyi számot megtaláltuk az indextömbben (5000-es szubrutin), a PN% tömbből kivesszük a relatív rekord sorszámot és beolvassuk a rekordot:

```
13200 PN=PN%(AP):GOSUB 2000:GOSUB 1000:REM REL. REK. POZ.
13210 INPUT#4,NE$,CI$,TE$:INPUT#4,DG$
13220 CLOSE 4
13230 OPEN 4,UN,3,"0:PATIENT":GOSUB 1000
```

Az adatokat kiírjuk a képernyőre:

```
13240 READ LI,O1,TX$,O2,MH
13250 DF$=NE$:GOSUB 32000 :REM MEZŐ KIÍRÁS : NÉV

13260 READ LI,O1,TX$,O2,MH
13270 DF$=CI$:GOSUB 32000 :REM CÍM

13280 READ LI,O1,TX$,O2,MH
13290 DF$=TE$:GOSUB 32000 :REM TELEFON

13300 READ LI,O1,TX$,O2,MH
13310 DF$=DG$:GOSUB 32000 :REM DIAGNOZIS

13320 IN FU<>4 THEN RETURN:REM HA A FUNKCIÓ NEM LEKÉRDEZÉS,
13320 :                   :REM AKKOR RETURN
13330 WAIT 198,15:GET A$  :REM BÁRMELYIK BILLENTYŰRE
13340 GOTO 13000          :REM VISSZA AZ ELEJÉRE


14000 REM
14010 REM PROGRAM VÉGE
14020 REM
14030 CLOSE 4:PRINT CHR$(147)
14040 GOSUB 4000          :REM INDEXFILE KIÍRÁS
14050 CLOSE 115:CLR:END
```

Figyeljük meg a relatív rekordot író, ill. olvasó utasítást:

```
10390 PRINT#4,NE$;",";CI$;",";TE$;",";CHR$(13);DG$

13210 INPUT#4,NE$,CI$,TE$:INPUT#4,DG$
```

Ha CHR$(13)-at nem írnánk ki, akkor az INPUT# STRING TOO LONG-gal elszállhatna, mert a rekord hosszabb lehet, mint az input puffer. Ha a vesszőt hagynánk el a CHR$(13) előtt, akkor, ha a rekordban nincs telefonszám, a TE$-ba olvasná a diagnózist.

A következő rutin a BASIC INPUT utasítást helyettesíti. A használt változók:

LI - kurzor pozíció: sor (1-25)
TX$ - szöveg (dialóg string)
01 - kurzor pozíció: szöveg oszlop pozíció (0-39)
02 - kurzor pozíció: bekérés kezdő oszlop pozíció (0-78)
MH - bekért mező hossza (02+MH<79)
DF$ - a rutin ebben adja vissza a bekért mezőt

```
31000 OPEN 63,3:REM KÉPERNYŐ FILE
31010 ? LEFT$(KL$,LI);TAB(01);TX$;TAB(02/1);":";TAB(02+MH);":";
31020 IF 02+MH>39 THEN MB=217+PEEK(214):POKE MB,PEEK(MB) AND 127
31030 PRINT LEFT$(KL$,LI);TAB(02);
31040 FOR I3=0 TO 10↑10  :REM VÉGTELEN CIKLUS
31050 : IF PEEK(211)<02 THEN POKE 211,02
31060 : IF PEEK(211)=02+MH THEN POKE 211,02+MH-1
31070 : POKE 207,0:POKE 205,1:POKE 204,0: REM KURZOR BEKAPCSOL
31080 : POKE 198,0:WAIT 198,15:GET A$   : REM BILLENTYŰRE VÁR
31090 : IF VK%(ASC(A$))=1 THEN POKE 204,1:PRINT A$;:NEXT
31090 : IF VK%(ASC(A$))=0 THEN 31080 : REM Érvénytelen billentyű
31100 : POKE 204,1:GET#63,B$
31105 : PRINT CHR$(157);B$;CHR$(157);A$;:REM CHR$(157)=KURZ.BAL
31110 : IF A$<>CHR$(13) THEN NEXT
31120 PRINT LEFT$(KL$,LI);TAB(02);
31130 INPUT#63,DF$
31140 FOR I3=MH TO 1 STEP -1
31150 : IF MID$(DF$,I3,1)=" " THEN NEXT
31160 DF$=LEFT$(DF$,I3)
31170 CLOSE 63:RETURN
```

A 31020-as és a 31070-es utasítások részletes magyarázata oldalakat tenne ki, ezért mellőzzük. A 211-es címen a kurzor oszlop pozíciója (a TAB értéke) található. A többször előforduló PRINT LEFT$(KL$,LI);TAB(02); utasítás a LI-ik sor 02 oszlopára pozícionálja a kurzort. A VK% tömb a megengedett billentyűket jelöli ki: elfogadható billentyűnél értéke <>0, ezen belül kurzor vezérlő billentyű >1, kiírandó karakter = 1:

```
200 DIM VK%(255)
210 FOR I0=ASC("0") TO ASC("9"):VK%(I0)=1:NEXT
220 FOR I0=ASC("A") TO ASC("Z"):VK%(I0)=1:NEXT
230 VK%(ASC(CHR$(157)))=2:REM KURZOR BALRA
240 VK%(ASC(CHR$(29))=2:REM KURZOR JOBBRA
245 VK%(13)=2          :REM CR
250 VK%(32)=1          :REM SZÓKÖZ
260 VK%(ASC("."))=1
```

Még két rutin hiányzik:

```
30000 REM
30010 REM NUMERIKUS MEZŐ ELLENŐRZÉSE
30020 REM
30030 NM=0:IF DF$="" THEN NM=1:RETURN
30040 FOR I3=1 TO LEN(DF$)
30050 : IF MID$(DF$,I3,1)=" " THEN NEXT:REM BAL OLDALI SZOKO-
30051 :                            REM ZÖK NEM SZÁMÍTANAK
30060 FOR I3=I3 TO LEN(DF$)
30070 : IF MID$(DF$,I3,1)<"0" OR MID$(DF$,I3,1)>"9"THEN RETURN
30080 NEXT
30090 NM=-1:RETURN

32000 REM
32010 REM EGY MEZŐ KIÍRÁSA
32020 REM
32030 PRINT LEFT$(KL$,LI);TAB(01);TX$;TAB(02);DF$:RETURN
```

A programnak így kell indulnia:

```
600 UN=8                    :REM KÉSZÜLÉK
610 OPEN 115,UN,15,"I":GOSUB 1000
800 GOSUB 3000
810 OPEN 4,UN,3,"0:PATIENT":GOSUB 1000
820 KL$=CHR$(19): FOR I=1 TO 24: KL$=KL$+CHR$(17):NEXT
999 GOTO 20000
```

A 610 és 800 sorok között ellenőrizhetjük a lemez azonosítóját, hasonlóan az adatlemezt előkészítő programhoz, ld. ott.

A DATA utasítások:

```
50000 DATA 4,1,"SZEMÉLYI SZÁM",16,11
50010 DATA 6.1,"NÉV",16,20
50020 DATA 8,1,"CÍM",10,40
50030 DATA 11,1,"TELEFON",16,6
50040 DATA 13,1,"DG.",10,40
50050 DATA 23,6,"OK (I/N)?",20,1
```

FELADAT-HARDWARE-SOFTWARE

Ebben a fejezetben felvetünk néhány gondolatot a mikrogép üzembe állításáról. A fejezetet azoknak ajánljuk, akik a gép beszerzéséről, a gépesítésről döntenek. (Ha könyvünk a vállalatnál először egy programozó kezébe kerülne, ezt a fejezetet olvastassa el főnökeivel.)

Ha még nincs gép

Ez a szerencsésebb eset. A konfiguráció kiválasztását a feladathoz igazíthatjuk.

Először is a feladat. Ha pl. a gépet a vállalat néhány ügyviteli folyamatának gépesítésére kívánjuk használni és már "letisztult", hogy mik is ezek, következhet a konfiguráció beszerzése. A konfiguráció:

- egy vagy több központi egység
- háttértároló(k)
- nyomtató(k)
- képernyő(k).

A konfiguráció összeállítását célszerű gyakorlott szakemberre bízni,de magunk is megpróbálhatjuk. Ha másra bízzuk, akkor sem kerülhetjük el, hogy olyan kérdésekre kelljen válaszolni, melyek pl. egy vállalat gazdasági vezetője számára nem megszokottak. De fontoljuk meg ezekre a kérdésekre adott választ, mert néhány nem kellően megfontolt válasz miatt esetleg új háttértárolókat kell vásárolnunk, vagy a drágán kidolgoztatott program-rendszert nem tudjuk használni.

Érzékeltetésül nézzünk erre egy egyszerű példát. Ez legyen a vállalat készlet nyilvántartásának és a forgalom figyelésének gépesítése. A területet igen jól ismerő gazdasági vezető és egy számítástechnikus helyzetfelmérő megbeszélése a feladatról:

a/ A vállalat gazdasági vezetője
- a vállalatnál n db raktár van
- a raktárakban a készletérték x mill. Ft
- a forgalom értéke y mill. Ft
- az anyagok elszámoló árának számítása...
- stb.

b/ A számítástechnikus kérdései
- Az egyes raktárakban hányféle anyagot tartanak nyilván?
- Az egyes anyagokról hány jelnyi információt tárolnak a mostani törzskartonon és ebből mennyi numerikus, mennyi karakteres?
- Hány tranzakciós tétel van havonta? (Nem bizonylat; kivét, bevét, stb., hanem tétel!)
- A tranzakciós tételek hány jelnyi olyan információt tartalmaznak, melyeket nem lehet a törzsállományból kimásolni?
- Stb.

Tehát nem könnyű a közös nyelvet megtalálni. Ha nem sikerül, a programrendszer felépítését (nem csak egyes programokat!) a megpróbált bevezetés után módosítani kell, esetleg újra tervezni az egészet.

A háttértárak. Pl. a COMMODORE SFD 1001 floppy ára kb. kétszerese a 1541-es floppynak. A kapacitása hatszoros, a sebesség is többszörös, a megbízhatósága sokkal nagyobb, az adathordozó költsége kb. 20% (egységnyi információra jutó költség).

A nyomtatók. A legolcsóbb mátrxinyomtató kb. 30 sor/perc sebességű, hangja hosszabb időn keresztül kibírhatatlan. A kétszer ennyibe kerülő nyomtató háromszor gyorsabb, programozható többféle betűtípus és íráskép, hangja kultúrált. A NAGYOBB TELJESÍTMÉNYŰ HÁTTÉRTÁR TÁROLÓK ÉS NYOMTATÓK RELATÍVE OLCSÓBBAK!

A konfiguráció legolcsóbb eleme a központi egység. Ha szükséges vegyünk többet belőle! (Nem jelent takarékosságot az olyan C 64, melyre 3 db 1541-es floppy, 1 8250-es floppy, egy mátrixnyomtató és egy plotter van kötve állandóan. Természetesen a különböző feldolgozásokhoz különböző konfiguráció kell, a többi periféria nem dolgozik.)

Ha már van gép

Ha a gépesítendő feladatokat meghatároztuk, a konfigurációt beszereztük, akkor "csak" a gépreszervezés, programozás van hátra. Lényegében az alábbi módszereket választhatjuk, esetleg kombinálhatjuk őket:

a/ Vásárolunk már megírt felhasználói rendszereket, amit általánosítva dolgoztak ki. Ezek többé-kevésbé eltérnek igényeinktől. Az eltéréseket két módon korrigálhatjuk. A rendszert (esetleg többszöri) módositással megpróbáljuk alkalmassá tenni igényeinknek megfelelően, vagy az ügyvitelt módosítjuk a rendszernek megfelelően. Az utóbbi általában kisebb költséggel jár, de nagyobb ellenkezést vált ki.

b/ A feladatot gyakorlott szakemberekre bízzuk, nincs saját programozói gárda. Ez a megoldás hozza a leggyorsabb és legjobb eredményt. Ha már a feladat megfogalmazásánál és a konfiguráció kiválasztásánál is bevontunk tapasztalt szakembert, a költségek sem lesznek túl nagyok. Talán itt van a legnagyobb szükség tapasztalt szakemberre. (Tizenéves gyermekünknek, aki elvégezte a TV-BASIC kurzust és ügyes programokat ír BASIC-ben, vagy kollegánk programozó matematika szakot most végzett lányának ne higgyük el, hogy gyakorlott szakember.)

c/ A gépre szervezést és programozást saját programozói gárdára bizzuk és a gyorsabb eredmény reményében beszerzünk olyan alapsoftware-t, melynél kevés programozással lehet boldogulni. Ezek adatbázis kezelő, vagy hasonló software-k (pl. SUPERBASE, MASTER, stb.). Az ilyen software-k általában bonyolultak, megismerésük hosszabb időt vesz igénybe, néhány program elkészítése miatt nem érdemes megtanulni őket. A megismerésük után viszonylag gyorsan lehet nagyobb rendszereket is késziteni velük és a rendszer könnyen változtatható. A módszernek két hátránya van. Az így készített rendszerek hatékonysága sebességben, háttértár kihasználásban elmarad az adott feladatra kidolgozott rendszerektől, ill. tartalmaznak olyan korlátokat, melyek nem minden esetben engedhetők meg (pl. az adatrekordok számának, az adatrekord hosszának, stb. korlátozása).

d/ Saját programozói gárdára bizzuk a programozást teljes egészében. A feladatokat egyedileg, adatbázis kezelő software nélkül oldjuk meg. Így lehet a gépi konfiguráció adta lehetőségeket maximálisan kihasználni. Ez a szemlélet nem jelenti azt, hogy nincs szükség a programozást segítő software beszerzésére (pl. HELP+, DBASIC, C 64 PLUS, stb.). De ezek a programozást segítő, nem helyettesitő software-k. Kezelésük egyszerű, az energiát saját rutinok, módszerek készítésére lehet forditani. Ez a "mindent magunk

csinálunk" módszer csak akkor lesz hasznos, ha több terület gépesítését határoztuk el, folyamatosan tudunk a programozóknak munkát adni. Ebben az esetben ez a legjobb módszer.

A programozókkal szemben némi türelemre van szükség. Hagyjuk, hogy pár hónapig ne készítsenek "éles" munkát. Ismerjék meg minél jobban a gépet, gyakorolják be a programozást segítő software-ket, írjanak saját rutinokat. Ez távlatban sokkal több hasznot hoz, mint az első héttől kezdve kiadott szoros határidejű munkák.

Még valami a gép használatával kapcsolatban. Ha már megvan a gép, ne zárjuk lakat alá, mondván nehogy elrontsák (azok, akik később dolgozni fognak vele). Ha eleinte minél többen hozzáférnek, próbálgatják (nem baj, ha játszanak), akkor az "éles" munka során nem fognak "félni" tőle, nem lesz ellenérzésük. Ez még akkor is megéri, ha a gép tényleg elromlik és javíttatni kell (csak a billentyűzet romolhat el. Ezt kb. 2000 Ft-ért kicserélik.).

"A" FÜGGELÉK

A képernyő kódok és ASCII kódok megfeleltetése:

KÉPERNYŐ KÓD	KARAKTER 1	KARAKTER 2	ASCII/CHR$ KÓD 1	ASCII/CHR$ KÓD 2
0	@	@	64	64
1	A	a	65	65
2	B	b	66	66
3	C	c	67	67
4	D	d	68	68
5	E	e	69	69
6	F	f	70	70
7	G	g	71	71
8	H	h	72	72
9	I	i	73	73
10	J	j	74	74
11	K	k	75	75
12	L	l	76	76
13	M	m	77	77
14	N	n	78	78
15	O	o	79	79
16	P	p	80	80
17	Q	q	81	81
18	R	r	82	82
19	S	s	83	83
20	T	t	84	84
21	U	u	85	85
22	V	v	86	86
23	W	w	87	87
24	X	x	88	88
25	Y	y	89	89
26	Z	z	90	90
27	[	[	91	91
28	£	£	92	92
29	]	]	93	93
30	↑	↑	94	94
31	←	←	95	95
32	szóköz		32	32
33	!	!	33	33
34	"	"	34	34
35	#	#	35	35
36	$	$	36	36
37	%	%	37	37
38	&	&	38	38
39	'	'	39	39
40	(	(	40	40
41	)	)	41	41
42	*	*	42	42
43	+	+	43	43
44	,	,	44	44
45	-	-	45	45

46	.	.	46	46
47	/	/	47	47
48	0	0	48	48
49	1	1	49	49
50	2	2	50	50
51	3	3	51	51
52	4	4	52	52
53	5	5	53	53
54	6	6	54	54
55	7	7	55	55
56	8	8	56	56
57	9	9	57	57
58	:	:	58	58
59	;	;	59	59
60	<	<	60	60
61	=	=	61	61
62	>	>	62	62
63	?	?	63	63
64	—	—	96	96
65	♠	A	97	65
66	|	B	98	66
67	—	C	99	67
68	—	D	100	68
69	—	E	101	69
70	—	F	102	70
71	|	G	103	71
72	|	H	104	72
73	╮	I	105	73
74	╰	J	106	74
75	╯	K	107	75
76	└	L	108	76
77	╲	M	109	77
78	╱	N	110	78
79	┌	O	111	79
80	┐	P	112	80
81	●	Q	113	81
82	_	R	114	82
83	♥	S	115	83
84	|	T	116	84
85	╭	U	117	85
86	╳	V	118	86
87	○	W	119	87
88	♣	X	120	88
89	|	Y	121	89
90	♦	Z	122	90
91	+	+	123	123
92	▒	▒	124	124
93	|	|	125	125
94	◢	※	126	166
95	◥	▒	127	—
96	szóköz		32	32
97	▌	▌	161	161
98	▄	▄	162	162
99	▁	▁	163	163

100	—	—	164	164
101	|	|	165	165
102	▒	▒	166	166
103	|	|	167	167
104	▒	▒	168	168
105	◤	▨	169	—
106	▌	▌	170	170
107	├	├	171	171
108	▖	▖	172	172
109	└	└	173	173
110	┐	┐	174	174
111	▬	▬	175	175
112	┌	┌	176	176
113	┴	┴	177	177
114	┬	┬	178	178
115	┤	┤	179	179
116	▏	▏	180	180
117	▎	▎	181	181
118	▍	▍	182	182
119	▔	▔	183	183
120	▀	▀	184	184
121	▃	▃	185	185
122	┘	✓	186	—
123	▖	▖	187	187
124	▘	▘	188	188
125	┘	┘	189	189
126	▝	▝	190	190
127	▚	▚	191	191

A képernyő kódok 128-255-ig a 0-127-ig ábrázolt karakterek inverzének felelnek meg.

A DOS ÜZENETEI

00, OK, 00, 00

Az utolsó I/O művelet hibamentésen zajlott le, ill. a hiba csatorna utolsó olvasása óta nem érkezett újabb I/O parancs.

01, FILES SCRATCHED, xx, 00

A SCRATCH parancs után adja ki.
xx - a törölt file-ok száma

20, READ ERROR, tt, ss

Olvasási hiba. A DISK-CONTROLLER nem találja a blokk fejet.
A diszketten valószinűleg fizikai hiba van.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

21, READ ERROR, tt, ss

Olvasási hiba. A DISK-CONTROLLER nem találja a szektor olvasásához a szinkronkaraktert.
A meghajtóban nincs lemez; vagy van, de nincs megformázva; az író/olvasó fej elállítódott; a diszketten fizikai hiba van.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

22, READ ERROR, tt, ss

Olvasási hiba.
Valószinűleg egy írás előzőleg hibás volt.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

23, READ ERROR, tt, ss

Olvasási hiba. Kontrollösszeg hiba van a szektorban. Egy vagy több adatbyte hibás.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

24, READ ERROR, tt, ss

Olvasási hiba. Egy vagy több byte dekódolhatatlan.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

25, WRITE ERROR, tt, ss

Írás hiba. Az írás utáni ellenőrző olvasás hibát mutat. A diszketten valószinűleg fizikai hiba van.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

26, WRITE PROTECT ON, tt, ss

A diszkett írásvédelmi ablaka le van ragasztva, de írni akartunk rá.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

27, READ ERROR, tt, ss

Ellenőrző összeg hiba a fejlécben.
A DISK-CONTROLLER nem olvassa be a pufferbe a hibás szektort.
A hiba egyik oka a gép földelésének elégtelensége lehet.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

28, WRITE ERROR, tt, ss

Írás hiba. Az adatblokk túl hosszú. (A DISK-CONTROLLER a következő blokk szinkronjelét nem érzékeli a megadott időn belül.)
A diszketten fizikai hiba van, vagy nincs jól megformázva.

tt - a hivatkozott sáv száma
ss - a hivatkozott szektor száma

30, SYNTAX ERROR

A parancscsatornán küldött parancsstringet a DOS nem tudja értelmezni.

31, SYNTAX ERROR

A küldött utasitás a DOS utasitás készletében nem szerepel.

32, SYNTAX ERROR

A küldött parancsstring túl hosszú. (40 karakter lehet.)

33, SYNTAX ERROR

Hibás file-név.
Az OPEN vagy SAVE utasításban a file-név hibás. Pl. a joker karakterek (?, *) használata helytelen.

34, SYNTAX ERROR

Hiányzik a file-név megadása.
Pl. a ":" hiányzik, ezért a file-név értelmezhetetlen.

39, SYNTAX ERROR

Érvénytelen parancs.
A parancs csatornán kiadott utasitás idézi elő.

50, RECORD NOT PRESENT

A rekord nem létezik.
A már beolvasott utolsó rekord után kiadott INPUT#, GET# utasítás idézi elő.
Relatív file esetén a pozícionáló parancs idézi elő. Ha írás miatt pozícionálunk, ez nem jelent tényleges hibát.

51, OVERFLOW IN RECORD

Túlcsordulás a rekordban.
Relatív file-ba íráskor lép fel. A PRINT# utasítással beépítendő string már túllépné a rekordhosszt. A felesleges karakterek "elvesznek".

52, FILE TOO LARGE

A file túl nagy.
A relatív file-ra kiadott pozícionáló utasítás végrehajtása túllépné a diszkette kapacitását.

60, WRITE FILE OPEN

Megnyitási kisérlet írásra olyan file-ra, ami nem volt lezárva.

61, FILE NOT OPEN

A file nincs megnyitva.
Egy olyan file-ra vonatkozó I/O utasítás váltja ki, mely nincs megnyitva.

62, FILE NOT FOUND

Egy olyan file-ra vonatkozó OPEN utasítás, vagy programra vonatkozó LOAD utasítás váltja ki, amely nincs a diszketten.

63, FILE EXISTS

A file már létezik.
A létrehozni kívánt nevű file már létezik a diszketten.

64, FILE TYPE MISMATCH

A file-típus hibás.
A file megnyitásakor - az utasításstringben kijelölt - a file típusa eltér a directoryban tárolt típustól. (Pl. létrehoztunk egy programfile-t, amelyet később relatív file-ként próbálunk megnyitni.)

65, NO BLOCK, tt, ss

Nincs blokk.
A Block-Allocate utasításnál keletkezik, ha olyan szektort akarunk lefoglalni a BAM-ban, amelyik már foglalt. Ha még van szabad szektor, a DOS lefoglalja és vissza adja a lefoglalt sáv és szektor számát.
Ha nincs már szabad szektor, tt=ss=0.

tt - a sáv száma amelyen a DOS a szektort lefoglalta
ss - a lefoglalt szektor száma

66, ILLEGAL TRACK OR SECTOR, tt, ss

Olyan számú sávra és/vagy szektorra hivatkozunk, amely nem létezik.

tt - a sáv száma
ss - a szektor száma

67, ILLEGAL SYSTEM T OR S

Egy szektorban a következő sáv/szektor mutató hibás.

tt - a sáv száma
ss - a szektor száma

70, NO CHANNEL

Nincs szabad csatorna.
Egy olyan I/O parancs váltja ki (pl. OPEN), amely további csatornát igényelne, de már az összes csatorna foglalt.

71, DIR ERROR

Directory hiba.
A diszketten lévő BAM nem egyezik meg a drive pufferben tárolttal.

72, DISK FULL

A diszketten már csak 3 szabad szektor van, vagy a directory betelt. (1541-es készüléknél 144 bejegyzés lehet.)

73, CBM DOS Vx.y

A DOS bekapcsolási üzenete. Ekkor nem jelent hibát. Ha olyan diszketten próbálunk írni/olvasni, amelynek formázási módja nem egyezik jelen készülékkel (más DOS verzióval történt), akkor hibát jelent.

74, DRIVE NOT READY

A meghajtóban nincs lemez, a készülék nincs bekapcsolva, vagy még nem érte el a szükséges fordulatszámot a diszkette.

75, FORMAT SPEED ERROR

A meghajtóra kiadott diszkette-formázási parancs végrehajtásánál a diszkette fordulatszáma nem jó. (Az eltérés 1% lehet.)
Az üzenetet csak a 8250-es drive adja.

76, CONTROLLER ERROR

A DISK-CONTROLLER-nél hardware hiba.

Kiadó: LSI Alkalmazástechnikai Tanácsadó Szolgálat
Felelős kiadó : Dr. Kovács Magda
Engedélyszám: 47537
ISBN: 963 592 541 7
Iv: 12,6 (A /5) iv
Forma: B/5

Készült:
PIREMON Nyomda, Debrecen
Felelős vezető: Dr Gere Kálmán
Munkaszám: 86/1479

intelorg

Az Önök vállalatánál mire használják a COMMODORE 64-et? Ismerik a gép lehetőségeit?

Manapság sokan úgy vélik, hazánkban a C 64-ek a szemétkosárba kerülnek, mielőtt egyáltalán bármire használták volna azokat.

A mikrogépekkel - s így a C 64-gyel - kapcsolatban többször felvetett kérdés, alkalmasak-e adatfeldolgozásra?
A kérdés így nem pontos, hiszen vannak olyan adatfeldolgozási feladatok, amelyeket ma a világ legnagyobb számítógépe sem tud elvégezni, s vannak olyanok is, amelyek a legkisebbel is könnyen megoldhatók. A kis gépek teljesítmény/ár aránya jóval felülmúlja a nagyobbakét.

A kérdés tehát úgy hangzik, milyen adatfeldolgozási feladatok megoldására alkalmas a C 64?

Ez a kör napról-napra bővül. Az Ön kezében már nem az a gép van, amit megvásárolt. A C 64 software fejlesztés eredményei MA MÁR valójában használhatóak - a gyakorlatban is!

Érdemes kihasználni az új lehetőségeket.

Büszkék vagyunk arra, hogy az INTELORG cég a fejlesztési munka élvonalában dolgozik. Az INTELORG software-ek döntő lépést jelentenek az adattárolási kapacitás növelése és a rendszerek áttekinthető, kényelmes kezelhetősége, ergonómiai szempontból is professzionális kivitele terén.

Ismerje meg a COMMODORE 64-et! Ismerje meg az INTELORG-ot!

ha commodore, akkor intelorg!

INTELORG GMK
Budapest, Dohány u. 58-62.
1077 Tel: 343-999

Ára: 119,-Ft